TESIS DOCTORAL

GRAMATICAS LOGICAS: RESOLUCION DE PROBLEMAS EN LENGUAJE NATURAL

por

Ana María GARCIA SERRANO
Licenciada con grado en Ciencias Matemáticas, sección Computación por la Universidad Complutense de Madrid

Presentado en la
FACULTAD DE INFORMATICA
de la
UNIVERSIDAD POLITECNICA DE MADRID

para la obtención del
Grado de Doctor en Informática

MADRID, DICIEMBRE DE 1987

TESIS DOCTORAL

GRAMATICAS LOGICAS:
RESOLUCION DE PROBLEMAS EN LENGUAJE NATURAL

que se presenta en la

FACULTAD DE INFORMATICA DE MADRID

para la obtención del grado de

DOCTOR EN INFORMATICA

Autor: Ana María García Serrano
Licenciada con grado en Ciencias Matemáticas
Sección Computación.

Director : José Cuena Bartolomé
Profesor Titular de Ciencias de la Computación
e Inteligencia Artificial de la Facultad
de Informática de Madrid.

TESIS DOCTORAL

GRAMATICAS LOGICAS:
RESOLUCION DE PROBLEMAS EN LENGUAJE NATURAL

Autor: ANA MARIA GARCIA SERRANO

Director: JOSE CUENA BARTOLOME

TRIBUNAL CALIFICADOR

PRESIDENTE: _____

VOCALES: _____

_____

_____

SECRETARIO: _____

Facultad de Informática, Universidad Politécnica de Madrid

Fecha:

PLANTEAMIENTO Y RESUMEN DE LA TESIS:

GRAMATICAS LOGICAS:
RESOLUCION DE PROBLEMAS EN LENGUAJE NATURAL

Esta tesis tiene por objeto estudiar las posibilidades de realizar en castellano tareas relativas a la resolución de problemas con sistemas basados en el conocimiento.

En los dos primeros capítulos se plantea un análisis de la trayectoria seguida por las técnicas de tratamiento del lenguaje natural, prestando especial interes a los formalismos lógicos para la comprensión del lenguaje.

Seguidamente, se plantea una valoración de la situación actual de los sistemas de tratamiento del lenguaje natural.

Finalmente, se presenta lo que constituye el núcleo de este trabajo, un sistema llamado Sirena, que permite realizar tareas de adquisición, comprensión, recuperación y explicación de conocimiento en castellano con sistemas basados en el conocimiento. Este sistema contiene un subconjunto del castellano amplio pero simple formalizado con una gramática lógica. El significado del conocimiento se basa en la lógica y ha sido implementado en el lenguaje de programación lógica Prolog II v2.

Palabras clave: Programación Lógica, Comprensión del Lenguaje Natural, Resolución de Problemas, Gramáticas Lógicas, Linguistica Computacional, Inteligencia Artificial.

ABSTRACT

LOGIC GRAMMARS: SOLVING PROBLEMS IN NATURAL LANGUAGE

The purpose of this thesis is to study the possibilities of performing in Spanish problem solving tasks with knowledge based systems.

We study the development of the techniques for natural language processing with a particular interest in the logical formalisms that have been used to understand natural languages.

Then, we present an evaluation of the current state of art in the field of natural language processing systems.

Finally, we introduce the main contribution of our work, Sirena a system that allows the adquisition, understanding, retrieval and explanation of knowledge in Spanish with knowledge based systems. Sirena can deal with a large, although simple, subset of Spanish. This subset has been formalised by means of a logic grammar and the meaning of knowledge is based on logic. Sirena has been implemented in the programming language Prolog II v2.

Keywords: Logic Programming, Understanding Natural Language, Problem Solving, Logic Grammars, Cumputational Linguistic, Artificial Intelligence.

A Manolo y Manuel

AGRADECIMIENTOS

Deseo expresar mi agradecimiento,

a José Cuena Bartolomé por dirigir y hacer posible esta tesis así como por introducirme y orientarme en las diversas parcelas de la Inteligencia Artificial,

a Robert Pasero en especial y a todos con los que he coincidido durante mis estancias en el Grupo de Inteligencia Artificial, GIA, de la Facultad de Ciencias de Marsella por su grata acogida, consejos e inestimable ayuda técnica,

al GLIA por su paciencia durante toda la etapa de realización de esta tesis y en particular de los actuales componentes,

a la Facultad de Informática de Madrid como ente y como grupo de personas por poner a mi disposición los medios con los que he contado para la realización de mi trabajo, y por aconsejarme o ayudarme en todo momento y en las mas dispares facetas,

a Martti Penttonen por sus valiosos comentarios,

a J. Cuena, J. Arteche, M. Muñoz, M. Rodriguez Artalejo, L. Laita y L. Ledesma por suplirme siempre que ha sido necesario en mis labores docentes,

y especialmente y con cariño a toda mi familia, que ha facilitado mi trabajo a lo largo de la realización de esta tesis asumiendo sin queja mis labores no profesionales.

INDICE

INDICE

CAPITULO I:
INTRODUCCION

I. INTRODUCCION

La comunicación entre la persona y el computador puede definirse como la evolución desde una situación centrada en el computador a otra centrada en la persona. En efecto, los primeros lenguajes eran practicamente una representación, apenas abstraida de la mecánica con la que se producían los cambios de estado en la memoria de la máquina, progresivamente, al ir evolucionando los computadores hacia un mayor rendimiento se planteó la necesidad de desarrollar aplicaciones mas complejas.
Sin embargo, la excesiva distancia entre la forma de entender los temas objeto de programación y la forma de computarlos daba lugar a excesivos errores y por tanto a una perdida de fiabilidad que se puso de relieve en la denominada crisis del software planteada a finales de los sesenta. Como resultado de esta crisis se planteó el movimiento de programación estructurada cuya idea motriz era la formulación de programas en base a un conjunto de estructuras de control y de datos análogos a conceptos intuitivos de la mente humana.
Puede afirmarse que desde primeros de los sesenta se produjo un sesgo en la concepción del software hasta entonces basado en una potenciación del concepto de la máquina y desde entonces orientado a la máxima adecuación con la forma de entender los problemas por las personas.

En esta línea cabe suscribir una parte de los proyectos de comprensión del lenguaje natural cuyas areas principales pueden considerarse:
- Traducción automatica. El objetivo de este area es traducir escritos en una lengua a otras lenguas manteniendo el significado.
- Comprension de documentos de forma que leido un documento sea posible transformarlo en un conjunto de informaciones representada con un cierto formalismo para que el sistema a base de comprender diversos documentos sea capaz de actuar

como lo haria una persona, realizar resumenes, responder a preguntas, etc..
- Generación de textos. Esta tarea consiste en transformar información codificada formalmente en información expresada en lenguaje natural (generación de informaes, de manuales de utilización etc.)
- Como parte de sistemas basados en el conocimiento orientado a la resolución de problemas. Para comunicación con un usuario en bases de datos o sistemas expertos, para control de equipos complejos, de enseñanza etc.

Los diferentes tipos de información necesarios en el proceso de comprensión del lenguaje natural pueden ser clasificados en:
- Conocimiento léxico de las palabras que forman parte de los dialogos.
- Conocimiento sintáctico que indica como agrupar las palabras en frases.
- Conocimiento semántico que permita la composición de los significados de las palabras de una frase para construir su significado.
- Conocimiento del marco de discurso que permita interpretar correctamente frases dentro del contexto conocido.
- Conocimiento de la situación mental de los participantes de un discurso: que tipo de objetivo tiene, como lo pueden conseguir etc.

La presente tesis se centra en la utilización de sistemas de comprensión del lenguaje natural en conexión con sistemas basados en el conocimiento de manera que se planteen como productos sistemas capaces de:
- Aceptar preguntas en lenguaje natural.
- Adquirir unidades cognitivas en lenguaje natural
- Producir respuestas y explicaciones en lenguaje natural.

En este tipo de sistemas cabria calificar su rendimiento

como próximo al de programación en lenguaje natural.

El sistema desarrollado realiza en un primer nivel estos objetivos supuesta una representación del conocimiento basada en reglas. Sus caracteristicas mas relevantes dentro del concepto integrado antedescrito son:

- Es un sistema que abarca un conjunto de frases en castellano suficientemente general para tratar con distintos sistemas basados en el conocimiento.
- Es incremental (diccionario dinámico, ampliación o modificación rápida en sus distintas funciones).
- El algoritmo que dirige el proceso de análisis de frases es óptimo y permite en expresar restricciones a verificar como la detección de ciertos errores gramáticales.

Este sistema se ha denominado Sirena (sistema de programamación en lenguaje natural).
Se ha realizado con el formalismo de las gramáticas lógicas, atrayente por su claridad y facilidad de modificación o ampliación. Además permite la integración de las fases de análisis y de conprensión (inferencia o deducción). El significado de las frases se representa con la lógica, sistema bien fundamentado.
Se ha programado, así como las aplicaciones realizadas con él utilizando el lenguaje de programación Prolog (Prolog II v2 de Marsella). Este lenguaje es el mas adecuado para el formalismo utilizado y presenta, entre otras ventajas la de ser transportable.

El sistema Sirena dispone de un conjunto de herramientas que permiten la aplicación del sistema a dominios variados con la misma efectividad. Un cambio de dominio implica la definición de palabras especificas y en pocos casos la introducción de nuevas estructuras gramaticales (ha sido verificado con la idea de que se utiliza para sistemas basados en el conocimiento en el que el conjunto de frases usuales no tiene necesidad de ser muy elaborado).

Ademas del tratamiento del castellano y una vez disponible el conocimiento relativo a un cierto problema, el sistema permite realizar deducciones a partir de preguntas del usuario con distintos motores de resolución. Al no ser el objeto de esta tesis, no se profundizará en ellos pero si se describiran brevemente. Una tarea tambien incluida en el sistema aunque se deja para trabajo furturo su modificación, para que sea realizada a forma análoga al análisis de una secuencia de palabras, es la generación de frases a partir de una formula o estructura semántica.

A modo de ejemplo se ha utilizado el sistema con:

a) Una base con conocimiento general de relaciones familiares. El análisis de las preguntas produce una formula en calculo de predicados pregunta a la base de conocimiento. Seguidamente se obtiene la respuesta con un módulo de deducción.

b) Una base de conocimiento general sobre la humedad del suelo con posibilidad de inundaciones. Los tipos de evaluaciones son confirmacion o peticion de evaluación. La semantica de concepto, atributo, valor permite obtener una formula asociada a la pregunta que es directamente evaluada con un sistema de reglas de producción.

c) Una base de conocimiento para evaluación de riesgos financieros. La formula asociada a una frase correcta en calculo de predicados es procesada con un módulo que la interpreta y evalua en el sistema.

El contenido de la presente tesis es el siguiente:

En el capítulo dos se muestra un panorama de técnicas clasicas de tratamiento de la comprensión del lenguaje natural. En la mayor parte de los sistemas presentados en este capítulo no hay una separación exacta entre las diferentes fases del análisis. Es por ello por lo que en la clasificación realizada puede ser posible la introducción de algunos métodos en varias categorias. Se ha elegido la que

representaba la caracteristica mas importante diferenciadora o innovadora en su momento.

En el capítulo tres se describen los distintos paradigmas que forman las Gramáticas Lógicas. Como se verá las distintas realizaciones basadas en las gramáticas de Metamorfosis son producto de una necesidad de mejorar la implementación o de captar ciertas regularidades del lenguaje.

En el capítulo cuatro se realiza una valoración de la situación actual y se proponen los objetivos y mejoras del sistema realizado.

En el capítulo cinco se describe la técnica del sistema propuesto (Sirena), el subconjunto del lenguaje implementado y la semantica utilizada. Así mismo se detallan las fases de las tareas que es posible realizar con el sistema.

En los distintos apendices se presenta en primer lugar un ejemplo completo de sesión con el sistema y finalmente los programas del sistema Sirena.

CAPITULO II:
TECNICAS CLASICAS DEL PROCESAMIENTO DEL LENGUAJE NATURAL

II TECNICAS CLASICAS DE PROCESAMIENTO DEL LENGUAJE NATURAL

En el procesamiento del lenguaje hay que realizar un estudio de sus componentes: léxico, morfológico, sintáctico, semántico y pragmático con técnicas en las que aparecen como puntos fundamentales los conceptos de gramática, análisis y representación del conocimiento.

En un sistema que procese el lenguaje natural pueden aparecer cada uno de los componentes o bien alguno de ellos siendo procesados en forma aislada o simultanea. El léxico del lenguaje o diccionario trata la etimologia, analogia o significado de las palabras que lo forman. La morfología es el estudio de las formas de las palabras. La sintaxis o estudio sintáctico se refiere a la coordinación y unión de palabras para formar frases. La semántica es el tratado del significado de las palabras. Finalmente la pragmática se refiere a los aspectos practicos del lenguaje.

La gramática es el conjunto de reglas que indican cuales son las frases del lenguaje. Análisis es el proceso de examinar cadenas de palabras a partir de la gramática con objeto de determinar la función de cada palabra en la cadena y crear cierto tipo de estructura representativa de la pertenencia de la frase al lenguaje. La representación del conocimiento del dominio (general o particular) sobre el que versan las frases puede ser realizado con distintos formalismos.

A continuación se presentan distintas técnicas de procesamiento del lenguaje natural con una descripción breve de los sistemas mas caracteristicos en cada una de ellas. La categorización se ha realizado a partir de una caracteristica diferenciadora pero se hace notar que algunos de los sistemas presentados aceptarian su inclusión en otro grupo.

II.1 PROCESO POR RECONOCIMIENTO DE PATRONES

La caracteristica principal de los primeros sistemas es que se interpretan las frases como un todo en vez de interpretarlas a partir de sus constituyentes. El método consiste en identificar "patterns" o patrones de palabras en la cadena de palabras de entrada. Cada patrón tiene asociada una interpretación que sera la de las frases que se identifiquen con él. En sistemas de este tipo mas evolucionados los patrones pueden estar formados con constituyentes o elementos semánticos, por lo que algunas partes de la interpretación comienza a ser constructiva, aunque sigue siendo un método fundamentalmente no constructivo y dirigido por el total.

Uno de los primeros sistemas fue **ELIZA** de Weizenbaum [Weizenbaum, 66]. Este sistema es una demostración de como con un mecanismo sencillo era posible aparentar un comportamiento sumamente inteligente. El sistema simula a un psiquiatra y el usuario es considerado su paciente.
Dada una frase de entrada se intenta unificar con alguno de los patrones y si esto es posible, se produce la respuesta con el patron de tipo respuesta asociado (en la cual podian aparecer partes de la frase de entrada por disponer los patrones de variables).

Eliza alcanza sus resultados con un bajo nivel de análisis de la frase e ignorando gran cantidad de información que aparece en ella. Para realizar un sistema complejo serian necesarios practicamente un patron modelo por cada una de las posibles frases del lenguaje. Un ejemplo de patrones en Eliza:

Modelo de pregunta	Conjunto de respuestas asociadas
A always B	- Can you think of a specific example? - When? - Really, always?

A you B me — Why do you think I B you?
— You like to think I B you, don't you?
— Souppose that I did B you. What would that mean?

A partir de este conjunto de patrones modelo de pregunta del paciente y sus respuestas asociadas se puede mantener la siguiente conversación:

P: People always stare at me.
E: Can you think of a specific example?
P: For instance you keep staring at me.
E: Why do you think I keep staring at you?
P: Because you hate me.
E: You like to think I hate you, don't you?

El programa **SIR** (Semantic Information Retrieval) diseñado por Shapiro, es de pregunta respuesta sobre una base de datos. Dispone de un conjunto de patrones con una acción asociada. Si la frase de entrada es asociada a uno de ellos entonces se ejecuta la acción asociada. Por ejemplo:

((X IS A Y !) (X Y) (UNIQUE GENERIC) (SETRS CAR CADR))
(DITTO (X Y) (UNIQUE UNIQUE) (EQUIV CAR CADR))

Así la frase "Charlie is a man" se corresponde con el primer patron haciendo que la variable X valga "Charlie" e Y "hombre". La función SETRS hace que "Charlie" pase a ser un objeto del conjunto "hombre". La sentencia "Charlie is Charles" establece una equivalencia entre ambos nombres con la segunda regla.

El sistema **STUDENT** [Bobrow, 1964] es de pregunta respuesta y resuelve problemas de algebra a nivel elemental. Opera reduciendo expresiones complejas a otras mas simples y transformando enunciados de problemas a un lenguaje de expresiones algebraicas con un conjunto de palabras clave y de reglas de transformación. Es limitado en el sentido de que cada información tiene una única representación.

Para refinar el tipo de análisis basandose en las mismas

técnicas, se definieron los patrones jerarquicos. En ellos sólo se unificaba una parte de la frase de entrada transformandola en un tipo de resultado canonico. Patrones de jerarquia superior se podian unificar con estas nuevas cadenas hasta llegar a la transformación de la frase de entrada en frase de respuesta.

El sistema mas conocido de patrones jerarquicos es **PARRY** de Colby [Colby, 77]. Modela a pacientes paranoicos.
Actua interpretando las frases de entrada como un todo e intentando unificarlas con casi 2000 patrones generales.
La frase de entrada es transformada a un conjunto de datos del modelo del estado mental de un paranoico, mas la representación de contenidos factuales de la entrada. Las respuestas se generan a partir del modelo completo y el contenido factual. El análisis de una frase de entrada sigue unos ocho pasos de "regularización". Por ejemplo: "A lend B a hand" ---> "A help B" transforma la frase "I want to lend you a hand" en "I help you".

La tercera variante consistia en realizar la unificación con patrones de elementos semánticos en vez de palabras. Este método era aún mas potente como se demostro con el sistema de traducción de ingles a frances de Wilks [Wilks, 75]. Este sistema se realiza con la teoría de **Preferencia Semántica.**
Este método ademas de potente permite eliminar cierto tipo de ambiguedades y recuperar errores gramaticales. Consiste en analizar primero la frase en ingles para obtener su representación interna, a partir de la que se genera la frase en frances. Los significados de las palabras se representan mediante fórmulas construidas con las mismas primitivas semánticas de los patrones. Por ejemplo el verbo "interrogate" tiene por significado ((man subj) (man obje) (tell force)) que significa que una persona fuerza a otra a decir algo.

En resumen, el análisis mediante "reconocimiento de

patrones" compara la frase o partes de la frase de entrada con patrones que tienen asociada una interpretación standard que pasa a ser la interpretación especificada de la frase, pero incluso para dominios muy restringidos el número de patrones modelo requerido es muy grande. Los sistemas que utilizan este tipo de análisis trabajan en un dominio limitado y en general tratan frases aisladas sin tener en cuenta el contexto. El sistema introduce muy poca teoria sobre el dominio de aplicación y suelen ser utilizados como interface con una base de datos.

II.1.1 TEORIA DE LA PREFERENCIA SEMANTICA

Wilks definío este método semántico de análisis que permite representar textos [Wilks 75,84] y realizar tereas de traducción de parrafos en inglés o francés.
Un texto en inglés se convierte primero a una representación semántica intermedia y posteriormente el texto es traducido. La representación semántica esta enfocada a la traducción y no a tareas como respuesta a preguntas u otras.

El mecanismo central consiste en identificar la frase a analizar con alguno de los patrones semánticos denominados "bare templates" utilizando una estrategia de "preferencia" entre las posibles soluciones. La preferencia se realiza conociendo las diferentes interpretaciones para diferentes dominios, tareas u otros conceptos.

La representación semántica se basa en un conjunto de primitivas semánticas de diferentes tipos (llamadas elementos) como son acciones, entidades, estados y cualidades acerca de lo que las personas comunican. Wilks ha identificado sesenta elementos que se agrupan en cinco clases. Por ejemplo a la clase entidad pertenecen los elementos MAN y STUFF y a la clase acción CAUSE Y FLOW. Estos elementos se utilizan para construir fórmulas (patrones) que representen el significado de una palabra. Por ejemplo el verbo "beber" será representado por:

```
((*ANI SUBJ)
 (((FLOW STUFF) OBJE)
  ((*ANI IN) (((THIS (*ANI (THRU PART))) TO)
                    (BE CAUSE)))
```

que debe entenderse como: beber es una acción (BE CAUSE), realizada por sujetos animados (*ANI SUBJ), sobre liquidos ((FLOW STUFF) OBJE), que produce que el líquido penetre en el objeto animado (*ANI IN), a traves de una abertura en el mismo ((THIS (*ANI (THRU PART))) TO).

Otro ejemplo es la fórmula semántica de representación del verbo "pegar" que significa: La persona sujeto causa el movimiento de una cosa con la idea de herir a un objeto animado.

((person subject) (animate object)
((to hit target) ((thing to move) causation)))

Con esta representación Wilks no pretende representar el significado de la palabra sino el sentido de la palabra dentro de una frase. Una palabra puede tener varios sentidos en una frase eligiendose aquel que hace mas coherente el sentido total de la frase. Por ejemplo la palabra vaso tiene el significado de cantidad de líquido y de objeto. En la frase "Manuel bebe un vaso de agua" se "preferirá" el primer significado por ser el verbo una acción de un objeto animado sobre un líquido. En el diccionario del sistema aparece para cada palabra cada uno de los esquemas de fórmulas asociadas que representan los distintos significados. Por ejemplo la palabra consejo (advise en inglés y conseiller en francés) será:

(ADVISE (CONSEILLER A (X FOLK MAN))
(CONSEILLER (X ACT STATE STUFF))

indicando que puede darse un consejo a un amigo (X FOLK MAN) o se aconseje algo (X ACT STATE STUFF).

Cada "bare template" es de la forma (actor, acción, objeto) ocupando cada una de las plazas un elemento, por ejemplo (MAN CAUSE MAN). Este conjunto fué determinado empiricamente y es facilmente modificable.

En el sistema dada una entrada se sustituye por cada una de las fórmulas asociadas a las palabras que la forman (construyendo todas las posibilidades). De esta forma puede haber varias secuencias de fórmulas para una única entrada. En este caso se reduce la secuencia de fórmulas a la secuencia de los elementos de cabeza de cada fórmula (una fórmula es una lista). A continuación la reducción se empareja con algun "bare template" (este emparejamiento se

realiza cuando haya tres elementos correlativos o no en la fórmula reducida que coincidan en orden y valor con los incluidos en la tripleta). Por ejemplo la fórmula reducida (KIND MAN HOW CAUSE KIND MAN) se empareja con el "bare template" (MAN CAUSE MAN).
Si el texto satisface varios entonces se aplica un algoritmo de expansión que selecciona aquella que satisface mas preferencias.
Finalmente el sistema aplica unas rutinas de generación de la traducción en base a la fórmula obtenida a partir del diccionario.

En los trabajos mas recientes de Wilks ha desarrollado un modelo en el que aparecen los "scripts" y los "frames" pero con un objetivo mas de representación del conocimiento que de análisis del lenguaje natural (no hay ningun sistema que lo utilice de momento).

II.2 PROCESO DIRIGIDO POR LA SINTAXIS

La sintaxis estudia las formas en que se pueden agrupar palabras para formar diferentes unidades de mas alto nivel (agrupar para conseguir estructuras de frases). El análisis sintáctico produce como estructura sintáctica, en general, un arbol llamado de derivación. El análisis se produce por aplicación de la gramática que define las frases que pertenecen al lenguaje que describe. Este método es constructivo, es decir la interpretación de una frase se construye a partir de las interpretaciones de sus constituyentes. El proceso del lenguaje comienza realizando el análisis sintáctico de la frase de entrada y a continuación el semántico o interpretación. Esta separación es bastante ineficiente y por ello hay analizadores dirigidos por la sintaxis en los que se efectua en forma paralela el análisis sintáctico y su interpretación.

Las gramáticas mas utilizadas en aspectos computacionales han sido las de contexto libre por ser faciles de definir y disponer de diversos algoritmos eficientes de análisis. Sin embargo no son adecuadas para representar ciertos aspectos de los lenguajes (concordancia entre palabras..) y no permiten expresar caracteristicas similares con reglas similares (frases en activa y pasiva exigen dos conjuntos de reglas distintos). Ademas no dispone de ningun mecanismo para detectar y eliminar ambiguedades sintácticas o semánticas.

Estos problemas intentaron ser resueltos por los linguistas. Entre ellos N. Chomsky [Chomsky 57, 65] con las **Gramaticas Transformacionales.**
La idea básica es la de utilizar elementos diferentes de las palabras que forman la frase a analizar, para la representación de la estructura de la frase. Estos elementos son las partes abstractas de la frase y forman la denominada "estructura profunda" (que posee la información que

transmite la frase) mientras que la secuencia de palabras que forman la frase se denomina "estructura superficial" (la que posee la frase que el sujeto emite). La relación entre ambas se determina mediante un conjunto de reglas llamadas reglas transfomacionales (que eran añadidas a las de la gramática). La gramática generaba un arbol sintáctico que era utilizado por una segunda fase de análisis con el conjunto de transformaciones sobre el arbol que ya obligaban a cumplirse las restricciones determidas en la primera fase.

Son una extensión de las de contexto libre ya que la estructura superficial es analizada con este tipo de gramática y la estructura profunda con el conjunto de reglas especiales, las reglas transformacionales. Este tipo de representación permite expresar de forma estandarizada la información significativa de la frase y dota de estructuras profundas similares a frases diferentes superficialmente. Sin embargo no permiten identificar frases con igual significado pero con diferente estructura superficial.

Segun N. Chomsky las gramáticas capaces de representar el conjunto de frases que pertenecen a un lenguaje natural han de generar tanto como analizar las frases del lenguaje asignando a cada una de ellas una descripción estructural. En el proceso de análisis o de generación se entiende bien diferenciada la parte semántica de la sintáctica.

Este método a pesar de expresar ya muchas carateristicas del lenguaje era computacionalmente mucho peor por lo que, aunque hubo intentos de realizar analizadores para estas gramáticas, no tuvieron gran exito dentro del procesamiento del lenguaje natural.

Otro tipo de análisis dentro de esta categoria es el desarrollado por D. Kaplan (1973), **GSP** (General Syntactic Processor). Es un sistema para generación y análisis de frases que puede emular al formalismo de análisis de una ATN.

Usa una sola estructura de datos "chart" para la gramática y la frase de entrada o generada. Es una estructura similar a la de arbol pero en el que los arcos se han reordenado para producir un arbol binario (como el método de Knuth, 1975) y los arcos se han transformado en nodos y los nodos en arcos. En estos arboles se distinguen dos tipos de relaciones entre nodos, (padre-hijo y de un nodo con el hermano a la derecha) que se expresan en forma explicita. La gramática se describe con una serie de nodos con transiciones entre ellos.

Woods [Woods, 70] desarrollo las Redes de Transicion aumentadas o **ATN** un método de expresar gramáticas computacionalmente aceptable y que seguia permitiendo tratar ciertas caracteristicas del lenguaje como lo hacia la gramática transformacional e incluso a veces en forma mas concisa. Consiste en una red recursiva (formalmente equivalente en potencia expresiva a las gramáticas de contexto libre) aumentada con un conjunto de tests para realizar a la entrada de los arcos y un conjunto de registros para almacenar información.

La primera realización operativa es **LUNAR** [Woods, 1972], sistema de pregunta-respuesta sobre la composición de minerales extraidos en la luna. El análisis es efectuado con una ATN y la representación interna se realiza con un lenguaje de acceso a la base de datos. El analizador no incluye informaciones semánticas produciendose por ello errores que a veces son recuperadaos en la fase de traducción al lenguaje de interpretación. Es muy potente y capaz de asignar un mismo resultado a diferentes frases con el mismo significado.

Otro sistema destacable es **SHRDLU** [Winograd, 1972] que representa un mundo de bloques de diferentes tamaños, colores y formas. Simula un robot que desplaza objetos. Es uno de los primeros que tratan simultaneamente el análisis sintáctico y semántico, referencias a una conversación previa,

representación de la información y resolución de problemas. Comienza a analizar hasta que encuentra una unidad con significado, en este momento se llama a rutinas semánticas. Si no encuentra problemas a nivel semántico continua con el análisis sintáctico y si las encuentra entonces hace que este comience de nuevo.

II.2.1 REDES DE TRANSICION AUMENTADAS

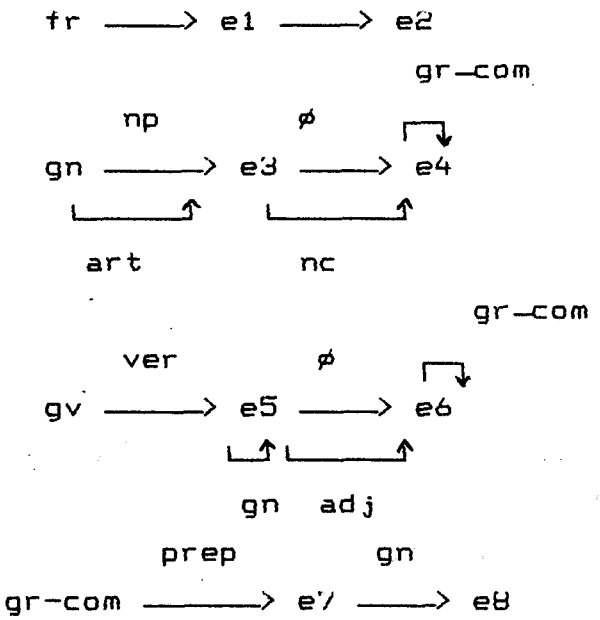
Las ATN fueron creadas por Woods y han sido ampliamente utilizadas (el primer sistema fue el Lunar, [Woods 72]. Es un formalismo mas potente que las redes de transición (son una extensión). Suelen estar implementadas en LISP aunque pueda ser utilizado cualquier otro lenguaje de programación. Esta elección es importante ya que en la definición tanto sintáctica como semántica de una ATN aparecen caracteristicas propias a las redes de transición y al lenguaje de programación.

Una red recursiva de transición es una colección de diagramas de transición, es decir de grafos orientados con estados (circulos dentro de los que hay una etiqueta de estado) y arcos etiquetados, un estado inicial (representados con doble circulo) y un conjunto de estados finales (se reconocen por la existencia de un arco de salida que no acaba en otro estado). Los diferentes diagramas representan los diferentes niveles de descripción de la gramática es decir a los distintos tipos de estructuras sintácticas. La etiqueta de los arcos indica el tipo de frase o de una parte de la frase de entrada o bien un simbolo del vocabulario no terminal que permita la transición por ese arco hacia el siguiente estado. Una frase de entrada es aceptada si comenzando por el estado inicial y tomando una secuencia de arcos se llega a un estado final.

Una red recursiva de transición difiere de los automatas finitos o redes de transición en que permite recursión al poder aparecer en los etiquetas simbolos no terminales en vez de sólo simbolos terminales. De esta forma el paso por un arco puede ser una llamada a una palabra de entrada, llamada o busqueda de un constituyente utilizando la red, comenzando por un estado determinado que cuando termina devuelbe el control al nivel anterior, o intento de hacer un pop en la pila. En la pila se va guardando la historia del análisis en

el sentido de que cada vez que se atraviesa un arco con un nombre de estado en su etiqueta, el estado actual al final del arco será guardado en la pila y el control saltará al estado de la etiqueta. Cuando se llega a un estado final se quita el estado de la cima de la pila y se le pasa el control.

Una gramática reconocida por una red recursiva de transición es debilmente equivalente a las gramáticas de contexto libre (reconocidas por automátas) ya que aunque reconoce el mismo lenguaje no dotan a las frases de la misma estructura (la de las gramáticas de contexto libre es mas compleja) y ademas no son capaces de generar arboles donde un nodo tiene infinitos hijos.
Un ejemplo de una red recursiva es:



Se extienden las redes recursivas para obtener las ATN's dotando a cada etiqueta de arco de una secuencia de condiciones y de un conjunto de acciones. Además cada red tiene un conjunto de registros asociados que sirven para almacenar informacion.

Una ATN construye una estructura, descripción de la frase al ir pasando de un estado a otro de la red de forma

que los elementos de la descripción parcial se van guardando en registros. Las condiciones que etiquetan a los arcos han de ser satisfechas para poder atrvesarlo y las acciones deben ser ejecutadas. Las condiciones tienen asociadas una función que calcula el valor a ser devuelto por la computación. Las acciones de construcción de estructuras utilizan los registros para ello. Estas acciones pueden ser de cambio en los contenidos de los registros en términos de su anterior contenido , del contenido de otros registros, del simbolo actual de entrada y/o del resultado de un nivel mas bajo decomputación. Ademas de guardar estructuras, los registros pueden almacenar otro tipo de información que pueda ser consultada por las condiciones de los arcos. Los estados finales tienen asociadas condiciones que deben ser satisfechas para producir una acción "pop". Con estos elementos las estructuras intermedias y la total se van construyendo con la información contenida en los registros y/o utilizando la palabra o conjunto de palabras de entrada ectuales y/o las las caracteristicas de las palabras anteriores (estas caracteristicas estan reflejadas en el diccionario).

Las gramáticas ATN y sus analizadores son independientes en cierta forma y por ello pueden adoptarse diferentes algoritmos de análisis. La mayoria utilizan una estrategia de análisis de izquierda a derecha, descendentes y con marcha atras, de tipo interprete o compilador.

Se describe a continuación un lenguaje para representar una ATN. Ha sido definido a partir del definido por Woods-70 y de las convenciones dadas por Bates- para la representación de una ATN.

<ATN> ::= (<conj-arcos> <conj-arcos>*)
<conj-arcos> ::= (<estado> <arco>*)
<arco> ::=
(CAT <cat> <test> <accion>* ((TO sig-est>)) /

```
(PUSH <estado> <test> <accion>* (<TO sig-est>))/ /
(WRD <palabra-ent> <test> <accion>* (<TO sig-est>)) /
(MEM <lis-pal-ent> <test> <accion>* (<TO sig-est>)) /
(VIR <tipo-consti> <test> <pre-accion>* <accion>*
                                    (<TO sig-est>)) /
(JUMP <sig-est> <test> <accion>*)
(POP <forma> <test>)
<accion> ::= (SETR <reg> <forma>) /
             (SETRQ <reg> <valor>) /
             (ADDL <reg> <forma>)
             (ADDR <reg> <forma>)
             (SENDR <reg> <forma>) /
             (SENDRQ <reg> <valor>) /
             (LIFTR <reg> <forma>) /
             (HOLD <tipo-consti> <forma>) /
             (VERIFY <forma>)
<forma> ::= (GETR <reg>) /
            (GETF <caracteristica>) /
            (BUILDQ <fragmento> <reg>*) /
            (QUOTE <estru-arb>) /
            LEX /
            x /
            (ABORT)
            (COND (<pred1> <e11> <e12>...<e1n>)
                  .....
                  (<predi> <ei1>...<eij>))
<fragmento> :: = + / # / * / @
<pred> ::= (NULLR <reg>) /
           (NIL) /
           (T) /
           (CHEKF <caracteristica> <valor>) /
           (CATCHEKF <pal-ent> <categ>) /
           (ENDOFSENTENCE) /
           (x-AGREE <forma> <forma>) /
           (x-START)
<estado> ::= <part1> "/" <part2>
<part1> ::= <tipo-consti>
```

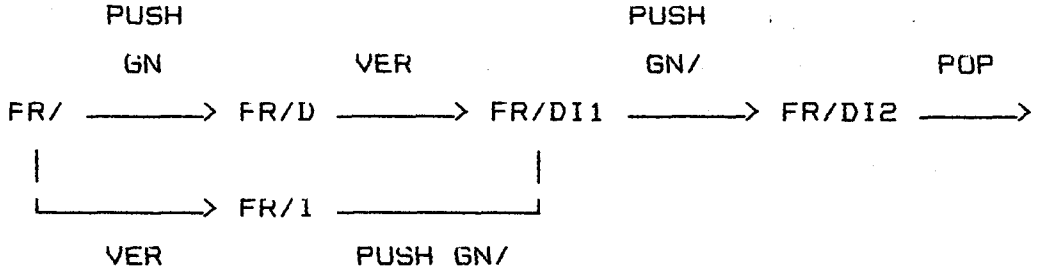
<part2> ::= <tipo-consti-sig> /
<caracteristicas-consti-ant>

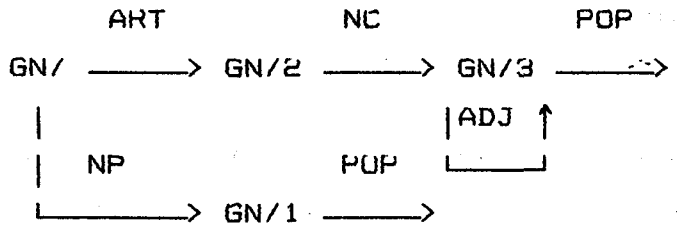
Un arco CAT se toma si la palabra actual pertenece a la categoria léxica <categ>. Los arcos CAT, WRD y MEM consumen palabras de la frase de entrada. El arco VIR mira si un elemento constituyente de un cierto tipo ha sido introducido por una acción HOLD realizada en algún arco anterior.

Cada vez que se realiza una acción del tipo PUSH , pueden ser iniciadas llamadas recursivas y su lista de registros (registros que contienen ciertas informaciones) se guarda en la pila mientras el analizador opera en el nuevo nivel de análisis que comienza con una lista de registros vacia. Cuando se realiza una acción de tipo POP la <forma> del arco tipo POP indica la estructura que debe ser devuelta como resultado del análisis del constituyente de una parte de la frase de entrada. Causa la vuelta del control al arco PUSH que produjo la llamada a este POP, recuperando la lista de registros del PUSH inmediatamente anterior.

Las acciones básicas de un ATN son SETR, SENDR y LIFTR, pero han sido ampliadas para simplificar la definición de las acciónes. Cuando despues de <reg> aparece una <forma> esta ha de ser evaluada y ese es el valor que pasa a tomar el registro. En LIFTR en <reg> se pone el valor de <forma> del nivel anterior al actual.
Un ejemplo:

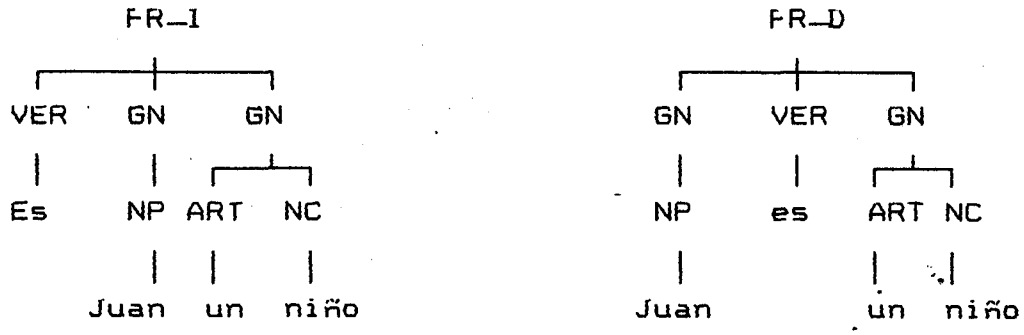
Registros= (FR-T, ADJ, GN, NP, VER, NC, ART)





Analisis de:

"Juan es un niño" y de "¿Es Juan un niño?"



La red ATN será en el lenguaje anterior de descripción:

```
(FR/ (PUSH GN T
     (SETR GN *) (SETR FR-T DEC)
     (TO FR/D))
     (CAT VER T
     (SETR VER *) (SETR FR-T INT)
     (TO FR/I)))
(FR/D (CAT VER T
      (SETR VER *
      (TO FR/DI1)))
(FR/I (PUSH GN T
      (SETR GN' *)
      (TO FR/DI1)))
(FR/DI (PUSH GN T
       (SETR GN *)
       (TO FR/DI2)))
(FR/DI2 (POP BUILDQ ( FR (+,+,(GV + +)) FR-T GN VER GN'))
(GN (CAT ART T
    (SETR ART *)
    (TO GN/2))
    (CAT NP T
```

```
    (SETR GN *)
    (TO GN/1)))
(GN/1 ( POP
(GN/2 (CAT NC T
      (SETR NC *)
      (TO GN/3)))
```

En [Woods 70] se encuentra el siguiente estudio de las ventajas que ofrece el uso de las ATN para el lenguaje natural.

Claridad. Las gramáticas de contexto libre permiten que los constituyentes de forman una categoria determinada puedan ser descritos directamente con las reglas. A pesar de que la gramática transformacional de Chomsky era muy potente para la descripción de las frases de un lenguaje, no tenia la caracteristica anterior ya que no es posible dada una regla saber facilmente en que tipos de construcciones va a poder ser utilizada. La ATN tiene el poder de la gramática transformacional y la claridad de las gramáticas de contexto libre.

Poder generativo. Las redes de transición aumentadas tienen mayor poder generativo que las gramáticas de contexto libre ya que estas para poder generar estructuras con un número ilimitado de constituyentes necesitarian un numero infinito de reglas. Las ATN pueden ser vistas como un formalismo que representa gramáticas de contexto libre con infinitas reglas de forma finita.

Otra caracteristica de las ATN es que no sacrifica su eficiencia para obtener potencia ya que consigue el poder de la gramática transformacional en poco mas tiempo que el necesario para un reconocimiento en contexto libre. Ademas es capaz de realizar lo equivalente a un análisis transformacional sin necesidad de separar la estructura superficial de la profunda.

Las gramáticas transformacionales son fundamentalmente generativas mientras que las ATN consideradas como hasta ahora son analizadoras pero si se considera que las

condiciones de los arcos son restricciones para la generación de las siguientes partes de la frase, adquieren el mismo poder generativo.

Eficiencia de la representación. Las ATN permiten la fusión de las partes comunes de reglas de contexto libre eliminando así procesos redundantes en el análisis o generación de frases y aprovechando información ya detectada por los tests en procesos posteriores. Esta información se almacena en los registros y produce un aumento de tiempo en procesos de análisis al tener que ser estudiadas las condiciones.

Captura de regularidades Siempre que en una gramática haya varias redes esencialmente copias de otro difiriendo sólo en alguna caracteristica indica que se está perdiendo regularidad. En las ATN estas caracteristiças pueden ser guardadas en los registros y utilizar solo una copia de la red.

Eficiencia de operación. Ademas de la fusión de partes comunes las ATN proporcionan mas eficiencia por permitir posponer decisiones (tomandolas sólo cuando hay elementos que la corroboran y no hasta entonces), permite reducir el no determinismo de la red así como la recursión salvo la inducida por elementos embebidos en si mismos. Ademas permite tomar decisiones sobre estrucutras de sentencias y posteriormente cambiarlas sin hacer una vuelta atras.

Flexibilidad para experimentación. El conjunto de operaciones básicas que pueden ser usadas sobre los arcos permiten el desarrollo de operaciones para el análisi del lenguaje natural a traves de la experiencia debida a la definición de la gramática. Permite además el uso y definición de diferentes representaciones de estructuras así como dejan introducir ciertos aspectos semánticos en las condiciones de los arcos e incluso construir representaciones semánticas.

Se han realizado muchas ATN para diferentes lenguajes, pero sigue presentando este formalismo las siguientes desventajas:

Complejidad y no modularidad: Aumentan segun aumenta la red. Es dificil aumentar o modificar elementos de la red sin tener que realizar gran cantidad de trabajo y produciendo a veces efectos no desados.

Fragilidad: La posición momentanea en el análisis en un elemento es demasiado determinante y por ejemplo si una palabra no es correcta gramaticalmente es muy dificil saber a que nodo podria ser adecuado saltar para continuar el análisis.

Ineficiencia en la busqueda con backtracking. En general el paso por una red ATN es no determinista y es necesario el uso de técnicas de busqueda. La forma natural de busqueda es el Backtracking, pero al no recordar los nodos fallo puede llegarse varias veces a un mismo tipo de nodo fallo.

Ineficiencia con respecto a los análisis con significado. Esto es debido a que se realiza un completo análisis sintáctico antes de comenzar la interpretación semántica, y por lo tanto pueden ser realizados muchos análisis sintácticos sin sentido antes de llegar al adecuado sobre todo si la gramática es ya compleja. Hay intentos actualmente de realizar sistemas de análisis que predigan en alguna forma que constituyentes no tendrian significado en futuros momentos.

Existe la posibilidad de construir ATN que acepten frases desde el punto de vista semántico. En general este tipo de redes necesitan más información que las sintácticas (mayor numero de redes de transición) y solo sirven para dominios concretos pero son muy eficaces.

II.3 PROCESO DIRIGIDO POR LA SEMANTICA

El análisis dirigido por la semántica exige la utilización de gramáticas semánticas, de forma que los constituyentes de las frases son constituyentes semánticos. Su fin era eliminar la produción de análisis sintácticos que obligaban a una posterior fase semántica ya que consideraban que aportaban poca información a la comprensión del lenguaje. sentido. Como las cracteristicas semánticas de los constituyentes estaban disponibles en el momento del análisis, este y la interpretación eran realizadas en el mismo momento. Esta técnica sin embargo solo funciona bien en dominios muy restringidos y donde sea posible clasificar objetos y relaciones concisamente.

Fue utilizado por primera vez por Burton para **SOPHIE** [Brow, Burton, 75], sistema para la enseñanza de la electronica. Contiene el conocimiento de forma declarativa (sobre circuitos electricos particulares) utilizando una red semántica y dispone de mecanismos que razonan sobre ese conocimiento.
La red semántica (ATN) define una correspondencia entre categorias sintácticas y conceptos del sistema (categorias semánticas) realizando los dos análisis simultaneamente.
En Sophie se categorizan todos los objetos y las acciones que necesita para analizar y conducir la comprensión a su dominio.

Otra aplicación fue el sistema **LADDER** [Sacerdoti, 77] sobre información de la Marina de los Estados Unidos y utilizando LIFER sistema definido por Hendrix [Hendrix, 77] para desarrollar gramáticas semánticas que permite construir directamente el significado de la frase evitando los problemas que aparecian con las ATN.
Lifer ademas permite producir las interpretaciones de las reglas que se usaron en el reconocimiento de la entrada obteniendo por ejemplo (como en Ladder) las interpretaciones

en terminos de comandos de consulta a la base de datos.

Con esta técnica es posible recuperar ciertas ambiguedades asi como resolver correctamente ciertos tipos de elipsis. Tambien permite introducir en ciertos momentos técnicas de pattern matching al ser posible definir categorias no exantamente sintácticas ni semánticas.

Entre las desventajas de este tipo de sistemas de análisis se encuentra la necesidad de realizar una nueva gramática para cada dominio por muy similar que sea a uno anterior, ademas tiende a ser grande rapidamente lo que provoca dificultades. Sin embargo esta demostrado que para dominios muy restrictivos es una técnica a tener en cuenta.

Como posibles soluciones a estos problemas se intentó en primer lugar realizar un análisis sintáctico pero en cuanto se determinara un constituyente se interpretaba. Es mas eficiente que los sistemas dirigidos por la sintaxis pero no como los dirigidos por la semántica y ademas no permiten introducir pattern matching. Un ejemplo es el sistema **RUS** en el que se aprecia la perdida de eficiencia con respecto a gramáticas puramente semánticas.

Otro intento consistió en construir un interface especifico para el dominio de la base de datos en cuestión. De esta forma conseguian buena eficiencia pero poca expresividad. Un ejemplo es el sistema **TEAM** de acceso a una base de datos relacional (se basa en una especie de relleno de plantillas con el vocabulario y caracteristicas morfologicas).

La tercera posibilidad fue combinar diversas técnicas de análisis de forma que construcciones o constituyentes no necesarios fueran eliminados de la formulación de la gramática semántica. Permiten el uso del pattern matching. El sistema **DYPAR** es un ejemplo de este tipo aunque ha habido

otras aplicaciones para acceso a bases de datos, para construcción de comandos de comunicación con sistemas expertos o interfaces para sistemas expertos.

Q-Seneca [Verdejo 80] es un sistema de pregunta respuesta en castellano definido con una ATN semántica. El analizador acepta una pregunta cuando es capaz de representarla en una red Seneca. Es un tipo de ATN en la que la estructura de la frase no se representa con un arbol de derivación. Una red Seneca se define con un conjunto de nodos, cada uno de un tipo (objeto, atributo, relación, clase o acciones abstractas o concretas), y con un conjunto de arcos etiquetados por relaciones (esun, elem,...). El modelo semántico del analizador es la red conceptual que modela el conocimiento del problema.
El subconjunto del castellano que reconoce el analizador esta formado por preguntas sobre objetos, propiedades o relaciones entre objetos. Es independiente del dominio.

La **Gramática de Grafos Conceptuales** ha sido desarrollada por Sowa [Sowa 84]. Los grafos conceptuales son un tipo de representación del conocimiento formado por simbolos (concepto), iconos (imagen genral) y percepciones (icono instanciado). Un grafo conceptual representa la estructura de conceptos y relaciones que expresan una proposición simple. Permiten realizar análisis y generación de lenguaje y es un formalismo útil para tratar ambiguedades o el ambito de cuantificadores.

II.4 TECNICA DE CASE-FRAME INSTANCIACION

El primero que utilizó las técnicas de instanciación con frames o casos para linguistica computacional fue Fillmore [Fillmore, 68] con las gramáticas de casos y a continuación Simmons, Schank y Riesbeck. Se considera una de las técnicas mas utilizadas actualmente ya que plantea bastantes ventajas al permitir combinar la busqueda bottom-up de constituyentes y de top-down instanciación de otros constituyentes.

Un case-frame consiste en un concepto principal y un conjunto de conceptos secundarios relacionados con el principal como lo determina un conjunto de reglas. Esta relación no es sintáctica sino semántica a pesar de la necesidad de ambas caracteristicas para transformar una frase en una estructura semántica case-frame (al principio se asociaba un case-frame con cada verbo del diccionario). En cada case frame hay tres tipos de conceptos asociados, los obligatorios (como por ejemplo el caso objeto en el frame asociado al verbo romper), los opcionales que aportan información y los prohibidos. Ademas asociado a cada case-frame existen indicadores posicionales o lexicales, indicando por ejemplo que tal concepto asociado al concepto principal ha de ser precedido por un conjunto determinado de palabras o bien ha de ser el que ocupa un cierto constituyente sintáctico. Ejemplo:

```
(comprar (case-frame
              agente: Manolo
              objeto: regalos
              instrumento:
              receptor:
              dirección:
              localidad: Zaragoza
              beneficiario: Alicia
              co-agente: )
         (modales
              tiempo: futuro
```

voz: activa))

Frame de: Manolo comprará regalos para Alicia en Zaragoza

Esta técnica difiere bastante de las otras ya que los casos de un Frame y los arboles de derivación no solo tienen representaciónes completamente diferentes sino que ademas utilizan distinto tipo de conocimiento para su construcción.

Las **Gramáticas de Casos** fueron introducidas por Fillmore [Fillmore 68]. La estructura profunda en estas gramáticas relaciona los elementos de una frase segun su actuación en relación con el verbo de la frase, disponiendose de un diccionario de "casos" donde se especifican los elementos requeridos por la aparición del verbo. Sin embargo Fillmore no pretendia que los casos fueran solo asociados a los verbos y ha existido una gran discusión sobre "que" debian ser [Bruce 75].
La estructura profunda no es un arbol sino un conjunto de argumentos para cada una de las acciones representadas (por ejemplo, la accion "dar" tiene por estructura profunda da(actor-da,actor-recibe,objeto)).

La **Dependencia Conceptual** es la teoria desarrollada por Schank para expresar el conocimiento contenido en una frase mediante la estructura de script [Schank 72] que permite describir situaciones (secuencias de acciones o hechos) estandar y puede ser visto como un patron de descripción. Toda conceptualización se expresa con una notación especial, la de dependencia conceptual, que reduce la representación en case-frames a la representación de acciones comunes como la composición de acciones primitivas mas estados intermedios y relaciones causales. De esta forma a partir de ciertas primitivas puede construir toda una red de relaciones entre ellas para expresar el significado de muchos tipos de estructuras. Es un intento de normalizar la representación semántica de las frases.
No es propiamente un sistema de análisis del lenguaje sino

mas bien de un método de compresión del lenguaje.

SAM, PAM [Schank, Abelson, 1977] son dos sistemas basados en esta técnica. SAM (script applier mechanism) es un sistema diseñado para guardar la información especifica de una familia en situaciones habituales como puede ser el acto de comer en un restaurante. Los scripts describen en forma estereotipada hechos y permiten obtener información elidida de la frase. PAM (plan applier mechanism) mantiene la idea de que la gente usa una serie de planes mentales para razonar. Construye scripts a partir de los actores y de los planes seguidos para realizar un objetivo.

ELI [Riesbeck, 1975] es un sistema en el que es la parte semántica la que dirige el proceso de análisis, con lo que la sintaxis aparece a nivel secundario. Cada palabra tiene unas precondiciones y una parte de acción asociadas, de forma que cuando se considera una nueva palabra en la fase de análisis se estudian sus precondiciones, si son satisfechas de borran de la "agenda" (lista de condiciones introducidas por la parte de palabras reconocidas hasta ese momento) y se añade a la parte de acción , es decir la estructura o condiciones a pedir a la siguiente palabra.

Otro sistema que utiliza esta técnica es **GUS** [Bobrow et all, 1977]. Su dominio de aplicación es el de agente de reservas de avion. Los frames contienen información sobre como puede ser un dialogo entre el agente y el comprador de billetes. Los programas semánticos de que dispone permite identificar hechos o lugares a los que se refiere el ususario.

POLITICS [Carbonell, 1978] simula las etapas de comprensión, razonamiento y respuesta de una persona en un dominio de hechos politicos aplicando la técnica de case-frame. Dado un conflicto politico y una ideologia, interpreta el hecho, predice nuevos hechos, responde a un conjunto de preguntas y

hace comentarios sobre la influencia de un conflicto en la politica de los EEUU y de la URS.

Las ventajas del análisis con case-frame pueden resumirse en las siguientes: Combinan el reconocimiento en bottom-up de los constituyentes del case-frame (conceptos) con el reconocimiento top-down de otros componentes semánticamente mas complejos. El trato diferente de los componentes de un case-frame permite en general, definir un algoritmo mas eficiente que detecte elipsis, y recupere ciertos errores. Combinan la sintaxis y la semántica. Finalmente es una representación adecuada para sistemas que trabajan con el conocimiento por no necesitar ningun tipo de transformación como lo necesitan las estructuras producidas por otros tipos de análisis.

Hay sistemas ya muy avanzados que contienen teorias explicitas sobre su conocimiento, planes y objetivos. Son útiles para sistemas basados en el conocimiento pues son capaces de analizar qué pretende el usuario y de decidir cómo responder al usuario a partir del conocimiento disponible.

TEIRESIAS [Davis 1977] realiza la tarea de adquisición de conocimiento en sistemas tipo Mycin. La explicación de los resultados obtenidos por inferencias del sistema Mycin se hace "fraseando" la secuencia de reglas que conducen al resultado. Puede ademas explicar el significado de una regla a varios niveles de detalle (con textos asociados).

PROSPECTOR [Duda, Gashing, Hart, 1979] es un sistema que diagnostica yacimientos geologicos. El conocimieto esta representado con redes semánticas y dispone de un programa de adquisisción de conocimiento. El conocimiento esta tipificado y para cada tipo el usuario ha de rellenar todas los puntos pedidas por el modelo. La gramática externa es facil de modificar para eliminar o introducir caracteristicas.

II.4.1 TEORIA DE LA DEPENDENCIA CONCEPTUAL

La teoría de la dependencia conceptual fue creada por Schank [Schank, 75] con objeto de disponer de un formalismo para la representación de conocimiento obtenido a partir del lenguaje natural y manejarlo para la comprensión de historias y textos. Se basa en que se puede comprender o actuar en una determinada situación si ha estado en ella enteriormente. La tarea de "entender" un texto es la construcción de una cadena conceptual que represente la conexión entre los hechos simples que aparecen en forma explicita así como hacer explicito lo que esta implicito.

Hay una serie de cadenas conceptuales prefijadas que representan secuencias de acciones bien conocidas por la generalidad y que comunmente son omitidas por ello, llamadas "scripts" (esquema mental de lo que son las cosas). Pretenden ser una respuesta a la pregunta ¿Como organiza la gente el conocimiento que posee sobre la vida cotidiana para comprender y encontrar adecuados ciertos comportamientos en situaciones particulares o actuar de acuerdo con la situación en que se encuentra?. Hay ciertas secuencias causales de hechos bien conocidas y que aparecen con frecuencia en la vida cotidiana en su totalidad o en parte por lo que resulta tedioso e innecesaria su construcción en cada momento. Son estas las situaciones que se almacenan en forma de scripts y son utilizados para recuperar información no aparecida en un texto o para confirmación de su corrección.

Para Schank estaba claro que la gente ademas puede adaptarse a situaciones en las que no tiene una experiencia previa y es para captar esta adaptabilidad para las que introduce la teoria la planificación. La capacidad de adaptación a una nueva situación viene determinada por la utilización de "conocimiento general" sobre conexión de hechos no conocidos explicitamente. Un plan es una secuencia de acciones a realizar que permiten conseguir un objetivo y

solo se utiliza cuando no es posible trabajar con scripts o cadenas causales. No se utilizan los planes desde el punto de vista de construcción sino como una subtarea de la comprensión de un texto.

Esta teoria de la representación del significado de las frases (y de textos) se basa en la presunción de que dos frases con igual significado han de tener idéntica representación en el modelo asi como en que las informaciones implícitas de las frases deben ser hechas explícitas en la representación de las frases. El elemento básico a partir del que se construye la teoria es el "concepto" o unidad mínima de información (proposición) categorizado en conceptos activos (actos, acción ...) y conceptos pasivos (objetos, estado ...).

Se trabaja con "acciones primitivas" para definir los conceptos de forma que puede haber conceptos que comparten acciones primitivas indicando su similaridad pero de forma que una cierta combinación de primitivas define un solo concepto, lo que indica la diferencia entre ellos.

CONCEPTOS ACTIVOS

Se define un conjunto de acciones primitivas que no son exactamente categorías de verbos que permiten definir conceptos activos:

MTRANS : transfiere una información mental a un animal o entre animales.

ATRANS : Transferencia de una relación abstracta como posesión, propiedad (dominio) o control.

PROPEL : Aplicación de una fuerza a un objeto

MOVE : Movimiento de una parte del cuerpo de un animal por él mismo

GRASP : Agarrar algo por un actuador.

DO : Indica una acción primitiva no conocida (la mayor parte de las veces se unifica con PROPEL)

Ejemplos:

aprender : nueva-información MTRANS LTM

decir : Persona MTRANS Persona

ver : ojos MTRANS PC

PC es un procesador consciente de que ofrece una información

recordar : LTM MTRANS PC

(LTM es un almacén de informaciones)

dar : ATRANS algo TO alguien

tomar : ATRANS algo To si-mismo

Se representa la frase "Juan empujó la mesa hasta la pared" como: Juan PROPEL mesa PTRANS pared

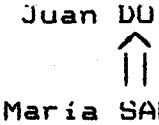
Este conjunto de acciones primitivas además simplifican las reglas de inferencia de afirmaciones a partir de acciones: si "información" MTRANS LTM entonces "información" es accesible

CONCEPTOS PASIVOS

Los conceptos pasivos son afirmaciones del tipo (atributo, valor) y pueden tener asociadas unas cotas para indicar el cambio de estado. Ejemplos :

CONCEPTO	VALORES	ESCALA
Salud	muerto, aceptable,...	(-10,10)
	Salud(-10) <=> Muerto	
Estado físico	muerto, herido, OK, ...	
Estado mental	loco, depresión, perfecto, ...	

Representación de las frases "Juan mató a María" y de "Juan lee un libro":



María SALUD (Muerto)

Juan MTRANS (información) TO LTM (Juan) FROM (libro)

inst (Juan ATTEND ojos TO libro)

CADENAS CAUSALES

En un texto la conexión entre frases es producida por relaciones causales entre ellas. Estas relaciones causales son de diferentes tipos. Es necesario disponer de un conjunto de reglas que nos permitan su identificación entre hechos conocidos se presente explicita o implicitamente. Ejemplo:
"Rosa llora, porque Manuel dijo que quería a Blanca"

(Manuel dijo que quería a Blanca) -> (Rosa llora)

"Manuel compró un camión. Cuando Alicia lo vió comenzó a jugar con el. Manuel lloró porque no pudo quitarselo"

(Manuel compró un camión) ->
((Alicia lo vió) ->
((Alicia comenzo a jugar con el camión) ->
((Manuel no pudo quitar el camión a Alicia) ->
(Manuel llora))))

Además son necesarias reglas que permitan tratar la incertidumbre que aparece por ejemplo en afirmaciones del tipo: (x produce y) pero no siempre.

Cuando un hecho produce un cambio de estado, aparecen de nuevo problemas de identificación de relaciones por lo que tambien son necesarias reglas:

"Juan se quemó la mano porque tocó el horno"

"Juan se quemó la mano porque olvidó que el horno estaba encendido"

La causa de quemarse en la primera frase es que "tocó el horno". Pero en la segunda un hecho mental: "se olvidó de que estaba encendido el horno", produce un hecho físico: "quemarse". Luego debe crearse una regla que infiera que un hecho intermedio físico: "tocar el horno". Luego hay que hacer una nueva regla para captar estos casos.
En general para poder asegurar que "x" causa "y" hay que comprobar si la causa es directa o hay hechos intermedios que producen "y".

El proceso de inferencia en esta teoría se realiza

directamente en la red de cadenas causales entre hechos simples.

TIPOS DE CADENAS CAUSALES

Hay dos aspectos a estudiar de la causalidad: la simple o sintáctica (fácilmente detectable y no destruible) y la compleja o semántica. En un mundo físico, para cada acción primitiva (hay 11) puede ser definida una lista de estados que la hacen posible, y una lista de estados a los que afecta su aparición. Se ve a continuación reglas que definen cadenas causales y que permiten realizar inferencias.

CS1: Ciertas acciones producen ciertos cambios de estado

(estado => act)

r

CS2: Ciertos estados hacen posible la aplicación de ciertas acciones.

(act => estado).

E

"María dió una patada a una pila de ladrillos" -->

"Juan se rompió una pierna"

Pasa a inferir con CS1 y CS2: "María dió una patada a una pila de ladrillos que entró en contacto con la pierna de Juan y se la rompió" ya que:

María PROPEL (María) TO ladrillos

pierna(Juan) ESTADO FISICO (herida)

Se tiene como reglas asociadas a PROPEL:

Rpropel1:

objeto1 CONTACTO FISICO objeto2

||r

algo PROPEL objeto1 TO objeto2

Rpropel2:

objeto PROPEL objeto2

||E

ESTADO FISICO (-)

Rpropel3:

objeto1 CONTACTO FISICO objeto2 (movible)

objeto1 PROPEL objeto2 TO objeto3

(algo PROPEL objeto1 TO objeto2)

Luego el ejemplo resulta:

María PROPEL ladrillos TO pierna(Juan)

María CONTACTO FIS ladrillos [Hip] ESTADO FIS pierna(Juan)

María PROPEL María TO ladrillos

CS3: Estados pueden destruir acciones

(Acción ==> Estado)

dE

CS4: Estados o actos pueden producir estados mentales

(Estado-mental ==> estado (o Acción))

I

CS5: Estados mentales pueden ser razones de acciones

(Acción ==> Estado-Mental)

R

Ejemplo:

"Juan previno a María de pegarla si se iba"

PROPEL (pegar)

PTRANS (permanece en la habitación)

(Luego la acción "pegar" aparece en el estado en que es falso que María permanece en la habitación).

Con todas estas reglas podemos decidir qué es y qué no es una cadena causal, se puede analizar correctamente la representación conceptual de palabras que contienen

implicitamente otras cadenas causales y se puede representar un texto.

Ademas en esta teoria la representación de las frases: "Juan dió un balon a Maria" y "Maria recibió de Juan un balon" no utiliza dos tipos diferentes de frames (uno para el verbo dar y otro para el recibir) sino que con la misma primitiva (frame) pero distinta cumplimentación se representa la idea comun a ambas frases de transferencia de posesión:

(ATRANS
rel: posesión
actor: Juan
objeto: balon
fuente: Juan
receptor:Maria)

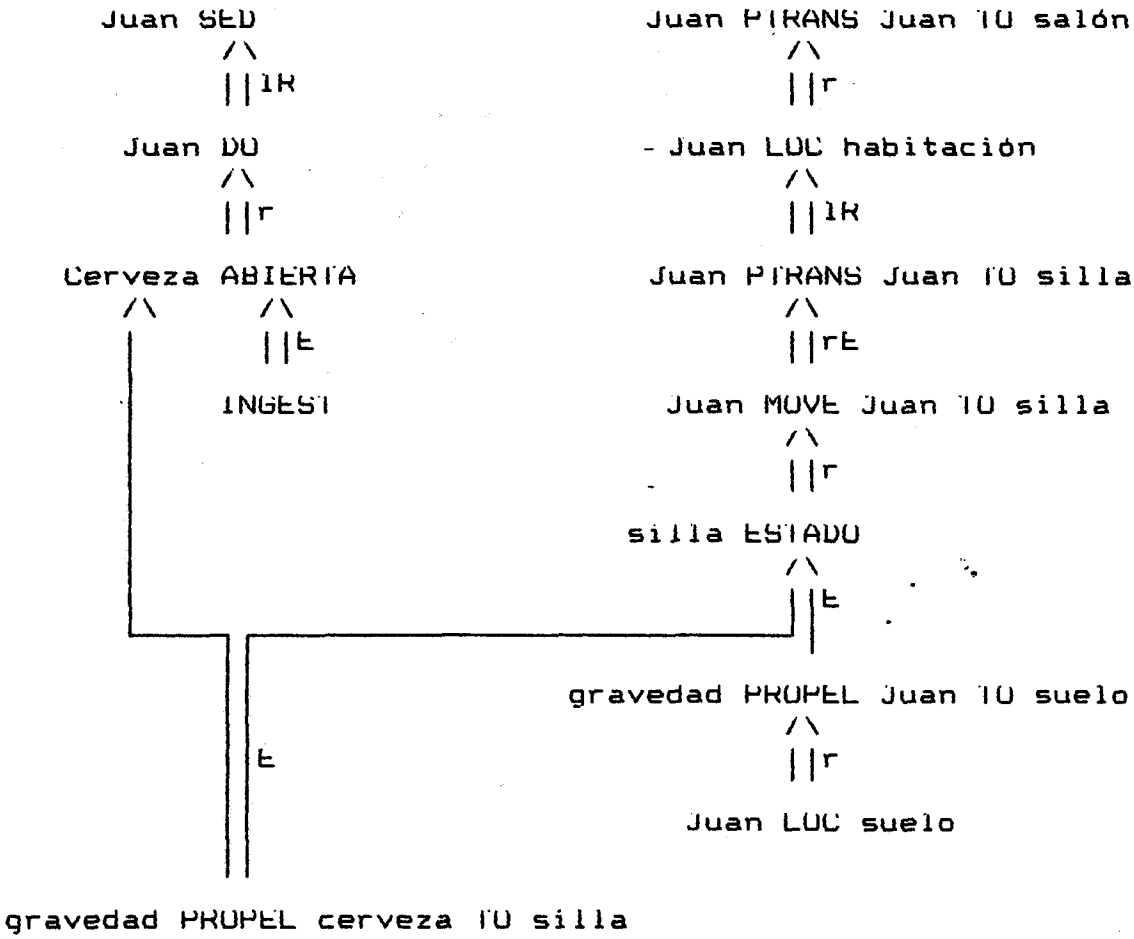
(ATRANS
rel: posesión
actor : Maria
objeto: balon
fuente: Juan
receptor:Maria

Estas dos estructuras obtenidas pueden ser emparejadas o comparadas y pueden ser utilizadas por reglas de inferencia asociadas a la primitiva ATRANS al tiempo de hacer el análisis de los verbos dar y recibir.

Un texto tiene sentido si es posible representarlo en una cadena causal y no lo tiene en caso contrario. Se muestra a continuación el ejemplo de Schank:

"Juan tenía sed. Abrió una lata de cerveza y fué al salón. Vió alli una silla nueva. Se sentó en ella. De pronto la silla se rompió y Juan se cayó al suelo. La cerveza se desparramó sobre la silla nueva."

La sed de Juan causa que se decida a realizar (DO) una acción que hace posible que la lata de cerveza esté abierta, lo que hace posible que sea bebida. La lata de cerveza abierta hace posible que se desparrame la cerveza sobre la silla.


Juan SED
/\
||IR
Juan DO
/\
||r
Cerveza ABIERTA
/\ /\
||E
INGEST
Juan PTRANS Juan TO salón
/\
||r
- Juan LOC habitación
/\
||IR
Juan PTRANS Juan TO silla
/\
||rE
Juan MOVE Juan TO silla
/\
||r
silla ESTADO
/\
||E
gravedad PROPEL Juan TO suelo
/\
||r
Juan LOC suelo
E
gravedad PROPEL cerveza TO silla

CAPITULO III:
FORMALISMOS LOGICOS PARA EL PROCESO DEL LENGUAJE NATURAL

III FORMALISMOS LOGICOS PARA EL PROCESO DEL LENGUAJE NATURAL

En paralelo con los desarrollos comentados en el capítulo anterior se ha ido creando, a partir de las ideas de A. Colmerauer, una línea de modelización integrada de sintaxis y semántica que si bien no cubre todas las necesidades de los modernos sistemas de bases de conocimiento, constituyen un vehiculo elegante y eficiente para realizar determinadas tareas en un subconjunto del lenguaje natural.

El primer trabajo fue la definición de los Sistemas-Q para traducción por A. Colmerauer. Sus realizaciones y estudios han marcado una línea de trabajo con gramáticas lógicas. Esta línea tiene como puntos principales la utilización de las gramáticas lógicas, definición semántica con formalismos lógicos y computación con lenguajes de programación lógica. Para todo ello se ha creado un conjunto de herramientas especiales como el lenguaje de programación Prolog que fue concebido en un primer momento con esta finalidad.

Las caracteristicas principales de distinción con las gramáticas o técnicas clásicas son:

1. Los simbolos no terminales pueden contener argumentos (no son sólo atomos como las gramáticas clásicas sino predicados con argumentos variables) con lo que:
   a. La reescritura se compone de unificación mas reemplazo.
   b. Es posible especificar y propagar restricciones contextuales mediante los argumentos, y así tratar reglas dependientes del contexto.
   c. Permiten asociar diferentes representaciones a las cadenas de entrada.
2. Se pueden introducir condiciones de aplicación en la parte derecha de una regla.
3. Representan a un conjunto infinito de reglas por permitir

variables, de forma que las reglas que utilizan en el momento de analizar sean instancias de ellas (solo es preciso expresar las meta-reglas de la gramáticas).

Las gramáticas lógicas son independientes del lenguaje de programación en que se implemente su interprete, pero el lenguaje Prolog es muy adecuado por ser la unificación una operación predefinida. Si ademas la estrategia de analisis es descendente, no determinista y en profundidad las reglas de la gramática pasan a ser ejecutables directamente por Prolog.

Han sido definidos diferentes tipos de gramáticas o métodos de análisis que permiten tratar ciertas peculiaridades de un lenguaje natural. Son una herramienta util y clara para analizar y sintetizar unidades del lenguaje y dotarlas de representaciones arbitrarias, como muestran las diferentes aplicaciones existentes realizadas en lenguajes como frances, ingles, castellano, portugues o finlandes.

Se presenta en este capitulo los diferentes tipos de gramáticas lógicas:

El primer trabajo que da lugar a todo este grupo de formalismos gramaticales fueron los Sistemas-Q [Colmerauer 70]. Un Sistema-Q es un formalismo de definición de operaciones sobre grafos que representan frases de un lenguaje y que permiten efectuar operaciones de sintesis y análisis del lenguaje. Las reglas que los definen tienen estructura analoga en ambas partes (secuencia de simbolos no terminales, terminales y condiciones).

Las gramáticas de metamorfosis [Colmerauer 75], son el descendiente directo de los Sistemas-Q. Las gramaticas de metamorfosis normalizadas (restricción estructural sin perdida de generalidad) tienen la parte izda de las reglas formada por un símbolo lógico no terminal seguido (o no) de simbolos lógicos terminales. Su parte derecha está formada

por una cadena cualquiera de simbolos lógicos terminales, no terminales o condiciones. A partir de ellas se han realizado variantes motivadas unas veces facilitar la implementación y otras para mayor generalización de las reglas dotandolas de un mayor poder expresivo.

Las gramáticas de clausulas [Pereira, Warren 80] son un caso particular de las anteriores pues en ellas la parte izda de la regla esta formada por un único simbolo lógico no terminal. Son directamente ejecutables en Prolog.

Las gramáticas de extraposición [Pereira 81] fueron propuestas con objeto de expresar más claramente las dependencias contextuales. La parte izquierda de las reglas está formada por un simbolo lógico no terminal, una secuencia de simbolos terminales o no terminales o la cadena "..." que parametriza simbolos lógicos y finalmente un simbolo no terminal. En la parte derecha aparece una secuencia cualquiera de simbolos acabada por la lista ordenada de cadenas "..." de la parte izquierda.

Las gramáticas de estructura modificada [Dahl, McCord 83] fueron creadas como una modificación de las de clausulas con objeto de tratar la coordinación.

Las gramáticas gapping [Dahl, Abramson 84] son una generalización de las gramáticas de extraposición que permiten que las cadenas incognita de la parte izda de la regla aparezcan en cualquier orden en la parte derecha.

Las gramáticas de puzzles [Sabatier 84] estan orientadas a ser una herramienta para los linguistas. Se dispone de unas construcciones fijas (elementos del puzzle) que pueden ser combinadas según un conjunto independiente de reglas estrategicas para construir estructuras mas complejas.

III.1 LOS SISTEMAS-Q

Fueron definidos por Colmerauer en el marco de un proyecto de traducción de frances a ingles y viceversa durante su estancia en el Departamento de Informática de la Universidad de Montreal e implementados con el lenguaje de programación FORTRAN.

Un sistema-Q es un conjunto de reglas que permiten hacer operaciones sobre los arcos de grafos (aristas orientadas) etiquetados por una cadena de forma que un conjunto de operaciones realizadas sobre el grafo puede dar lugar a un análisis o síntesis de la estructura formada por las etiquetas de los arcos.

La característica fundamental de estos sistemas es que pueden realizar el análisis o síntesis de frases como operaciones formales sobre grafos orientados gracias a que la estructura de la parte derecha e izquierda de la regla son análogas. Otra característica es la posibilidad de encadenamiento entre varios Sistemas-Q de forma que cada sistema implemente una fase del análisis. En general, una frase en ingles no pasa por menos de 15 Sistemas-Q hasta ser traducida al frances. La depuración de estos sistemas es simple al permitir la síntesis del lenguaje. Ademas la aparición del lenguaje Prolog facilitó su implementación.

DEFINICIONES BASICAS

Arbol : es un elemento de base formado por nodos marcados por etiquetas (secuencia de letras, números o símbolos especiales), parentesis o comas. Ejem:
FRASE(GN(NIL),GV(V(COME)))

Lista : es una secuencia de árboles separados por comas (NIL es la lista vacía). Ejem: GN(NIL),GV(V(COME)). Sobre las listas se definen las operaciones =, <>, DANS, HORS.

Arbol (lista) parametrizado : es un árbol (lista) en el que

aparecen eventualmente incógnitas que podrán ser reemplazadas por un árbol, una etiqueta o una lista. Ejem: FRASE(GN(x),GV(y)) con x e y incognitas. Un árbol (o lista) parametrizado representa el conjunto de todos los árboles (listas) que pueden ser obtenidos sustituyendo cada parámetro de tipo etiqueta por etiquetas, de tipo árbol por árboles y de tipo lista por listas (sustituir es reemplazar las diferentes apariciones de un parámetro por el mismo elemento). Un arbol es un caso particular de las listas.

Cadena : es una secuencia de árboles separados por el signo "+". Ejem: GN(Pedro) + GV(come). Operación sobre las cadenas: sustituir una subcadena por otra subcadena. Una cadena parametrizada es una cadena de arboles parametrizados.

Grafo de árboles: es un conjunto de arcos etiquetados por árboles que unen un conjunto de vertices. Verifica las condiciones siguientes : 1. No tiene circuitos. 2. Existen dos vértices distinguidos: el de entrada y el de salida (y sólo dos). Se representa a los vértices con enteros no negativos y a los arcos con la secuencia, número etiqueta número. Ejem: 1 GN(Pedro) 2

Camino en un grafo: es una secuencia no vacía y finita de arcos $f_1, f_2, \ldots, f_n$ / n=1 ó n > 1 y paratodo i, f_i llega al nodo del que parte f_{i+1}. Sea ai el árbol que etiqueta f_i. Entonces ese camino está etiquetado por $a_1+\ldots+a_n$.

DEFINICION DE SISTEMA-Q

Un Sistema-Q Simplificado (SQS) o normalizado está formado por un conjunto de reglas de la forma:

$a_1+\ldots+a_p == b_1+\ldots+b_q$ (== representa "se reescribe en")

donde los ai y bj son árboles cualesquiera.

estos conjuntos de reglas permiten transformar un grafo etiquetado en otro mediante las siguientes operaciones:

1. Adición de arcos: Para todo camino $a_1+ \ldots +a_p$ en un grafo si existe una regla $a_1+ \ldots +a_p == b_1+ \ldots +b_q$ se añade en

el grafo el camino $b_1 \ldots b_q$ con p=q.

Continuando este proceso se llega a que no se puede aplicar más reglas (resultado independiente del orden de aplicación de las reglas) o a que el resultado sea indefinido si existe recursividad.

2. Supresión de arcos: Se suprimen aquellos que no figuran en el camino parte derecha de una regla pero que sí aparecen en el camino parte izquierda de una regla.

Un Sistema-Q es una secuencia de reglas y de comentarios:

1. Un comentario es un texto con un indicador de texto.
2. Una regla está formada por una parte izquierda,un simbolo "==", una parte derecha, un simbolo "\" y una condición. Las partes derecha e izquierda son cadenas parametrizadas y las condiciones expresiones booleanas. Si una regla tiene una parte identica a la de otra regla anterior (en la parte izda o en la derecha) puede ser sustituida la parte comun en la segunda regla por un indicador "--" a la regla anterior.

Los Sistemas-Q necesitaban ser simplificados para su implementación hasta la aparición del lenguaje Prolog. Para ello se realizan las cuatro eliminaciones siguientes;
Eliminación de comentarios.
Eliminacion de los indicadores de relación entre reglas. Dada la regla en orden, se reemplaza cada indicador por la parte derecha de la regla precedente si está en la parte derecha (análogamente, izquierda) o por la condición, si aparece en la condición. Luego la primera regla no puede tener este tipo de indicadores y una regla con condición que contiene un indicador no puede ser precedida por una regla sin condición.
Eliminación de parámetros. Se reemplaza cada regla que contiene parámetros por todas las reglas obtenidas al sustituir esos parámetros por los elementos adecuados

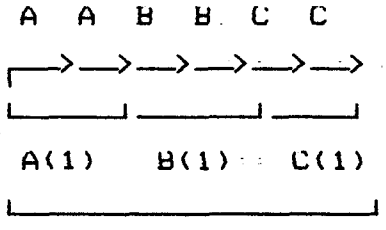
(listas, etiquetas o arboles cualesquiera). En el conjunto final de reglas, todas serán distintas.
Eliminación de condiciones. Se suprimen las reglas en las que la condición no sea satisfecha y se eliminan las condiciones de las reglas que sí las verifican.

Por razones de eficacia se pide que todo parámetro del miembro derecho o de la condición de la regla ha de aparecer en la parte izquierda. Ejemplo:

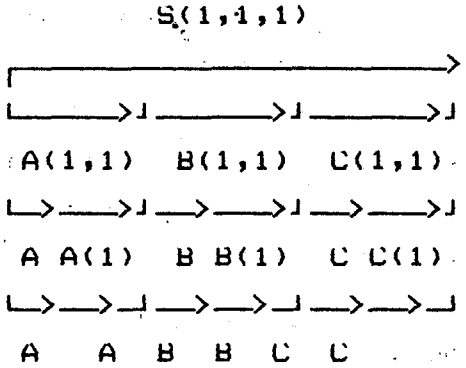
Sistema-Q = { x + x(y) == x(1,y) \ x DANS {A,B,C}
A(x)+B(x)+C(x) = S(1,x)}

Sistema-Q simplificado =
{ A+A==A(1) A+A(1)==A(1,1) ...
B+B==B(1) ...
C+C==C(1) ...
A+B+C==S(1) A(1)+B(1)+C(1)==S(1,1) ...}

Aplicación de las reglas a la cadena AABBCC:



La aplicación inversa de las reglas:



III.2 GRAMATICAS DE METAMORFOSIS

Las gramáticas de metamorfosis son el formalismo sintactico introducido por Colmerauer en 1975 a partir de los Sistemas-Q y de las técnicas de deducción automática.

Estas gramáticas, son una generalización de las de contexto libre, permiten análisis y síntesis de frases del lenguaje que representan y, por primera vez, manipulan estructuras complejas y generales como son los árboles.
Con ellas es posible definir gramáticas de contexto libre mediante reglas con estructura de cláusula de Horn de forma que las componentes (elementos terminales y no terminales del altabeto de la gramática) de cada regla pueden tener "argumentos" (en el sentido del cálculo de predicados) que permitirían representar ciertas restricciones o expresar cierta información.

La primera implementación fue realizada por V. Dahl, en su tesis de tercer ciclo [Dahl 77] (con la primera versión del Prolog creado por el GIA Grupo de Inteligencia Artificial de Marsella) utilizando una gramática de metamorfosis normalizada para acceso a una base de datos.

DEFINICIONES BASICAS

Sea F un conjunto de símbolos funcionales y un conjunto finito de variables. Un término en F se define:

1. Si v_i es variable entonces v_i es término.
2. Si $f \in F$ con orden cero, entonces f es un término.
3. Si $f \in F$ con orden n y $t_1 \ldots t_n$ son términos entonces $f(t_1 \ldots t_n)$ es un término.

H(F) o sólo H denota el conjunto de términos que no contienen variables (Universo de Herbrand).
Sea F un conjunto de símbolos funcionales que contiene al símbolo funcional "." de orden dos (de construcción de la estructura de lista) y a "nil" de orden cero (indicador de la

lista vacia).
Sea V un subconjunto de H llamado vocabulario.
Una cadena de longitud n en el vocabulario V es un término de la forma:

$a_1.a_2. \ldots .a_n.nil$ con $n>=0$, $a_i \in V$ (nil de longitud 0)

V^* es el conjunto de todas las cadenas.
Notación : ä denota la cadena de longitud uno a.nil
Ley de composición interna sobre V^*: la concatenación definida por:

1. si x=nil, entonces x.y=y
2. si $x=a_1 \ldots a_n.nil$, entonces $x.y = a_1..a_n.y$

Sea -> una relación binaria sobre H y sea V el vocabulario sin variables tal que $V \subset H$.
La relación -> es de reescritura en V^* sii

paratodo $x,y \in H$, si $x \rightarrow y$ entonces $x,y \in V^*$

Relaciones de reescritura entre elementos de H:
$a \rightarrow^0 b$ sii a=b y $a,b \in V^*$
$a \rightarrow^{i+1} b$ sii existen $u,v,r,s \in V^*$ tal que $a=urv$, $r \rightarrow s$ y $usv \rightarrow^i b$
$a \rightarrow^* b$ sii existe $i>=0$ tal que $a \rightarrow^i b$

DEFINICION DE GRAMATICAS DE METAMORFOSIS

Una gramática de metamorfosis G se define con una quintupla (F, V_t, V_n, V_s, ->) donde:

1. F es un conjunto de símbolos funcionales conteniendo "." y "nil".
2. V_t es el vocabulario terminal con $V_t \subset H$
3. V_n es el vocabulario no terminal con $V_n \subset H$
   Hipótesis $V_n / \backslash V_t = \emptyset$ y $V=(V_t \cup V_n)$.
4. $V_s \subset V_n$, donde V_s son los símbolos iniciales
5. -> es una relación de reescritura en V^* con la restricción de que si $x \rightarrow y$ entonces x =/= nil

L(G) es el lenguaje generado por G gramática de metamorfosis.

$L(G) = \{ t \in Vt^* \mid$ existe $a \in Vs$, $a \in Vs$, ä $\rightarrow^* t \}$

Si $a \in V_s$, $t \in V_t^*$ y $ä \rightarrow^* t$, a es la estructura profunda de t.

Ejemplo:

F = {nil, a, b, <, >, +, fin, formula, valor, . }

con ar(0) , ar(1) , ar(2)

Vt = {a, b, <, >, + }

Vn = Vs U {fin} U {valor(x) | x∈H}

Vs = {formula(x) | x∈H}

Reglas = {formula(a).nil -> a.nil
formula(b).nil -> b.nil
formula(x).nil -> valor(x).nil
valor(y.z).nil -> <.nil formula(y).nil
fin.nil. valor(z).nil
valor(nil).nil -> nil.nil
fin.nil <.nil -> +.nil
fin.nil -> >.nil
donde y,z∈H(F), x=u.v∈H(F)

En fase generativa:

formula(a.b.a.nil).nil -> valor(a.b.a.nil).nil
-> <.nil forumla(a).nil fin.nil
valor(b.a.nil).nil
-> <.nil a.nil fin.nil valor(b.a.nil).nil
-> <.nil a.nil fin.nil <.nil formula(b).nil
fin.nil valor(a.nil).nil
-> <.nil a.nil +.nil b.nil fin.nil valor(a.nil).nil
-> ...
-> <.nil a.nil +.nil b.nil +.nil a.nil >.nil

luego se ha generado la palabra <a+b+a> con estructura profunda (a.b.a.nil).nil

GRAMATICA DE METAMORFOSIS EN FORMA NORMAL

Una gramática de metamorfosis está en forma normal si se verifica que si ä x->y entonces a∈V_n y x∈V_t^*. Es decir la regla tiene en la parte izda una cadena que comienza por un único no terminal seguido o no por una secuencia de simbolos terminales.

Para toda gramática de metamorfosis G=(F,V_t,V_n,V_s,->)

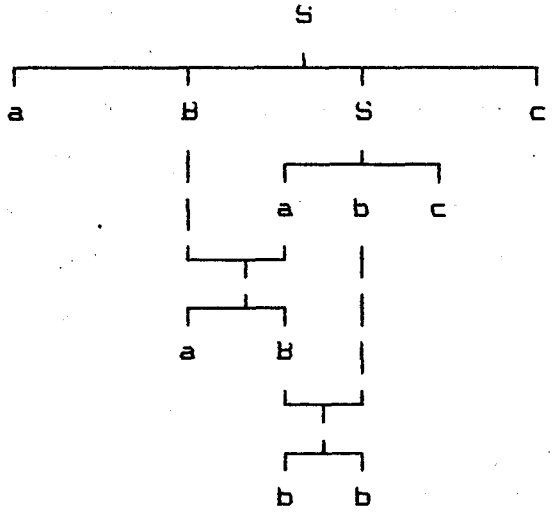
existe una gramática $G'=(F',V_t',V_n',V_s',->')$ de metamorfosis en forma normal tal que para cada $a \in V_n$ y $t \in V_t^*$, $ä ->^* t$ sii $ä ->'^* t$. Luego no se pierde generalidad por trabajar con una gramática en forma normal.

Otra propiedad de estas gramáticas es que Paratodo $t \in V_t^*$, paratodo $x,y \in V^*$ y paratodo $i>=0$ si $tx ->^i$ y entonces existe $z \in V^*$ con $x ->^i z$ y $tz=y$.

Ejemplo de gramatica de metamorfosis del lenguaje $\{a^n b^n c^n\}$:

$\{S -> aBSc,\ S -> abc,\ Ba -> aB,\ Bb -> bb\}$

que analiza:



III.3 GRAMATICAS DE CLAUSULAS

Fueron definidas por Pereira y Warren [Pereira, Warren 80] y son equivalentes a las gramáticas de metamorfosis en forma normal. Es una extensión de las gramáticas de contexto libre que constituye en si misma una descripción del lenguaje y un procedimiento efectivo para implementación de sistemas de tratamiento del lenguaje.

Una regla en estas gramáticas es una clausula de Horn (conjunción de predicados implica un solo consecuente), es decir en la parte izquierda de la regla debe aparecer un único simbolo no terminal. Analizar una frase del lenguaje se transforma en el problema de probar que un "teorema" (frase) es deducible del conjunto de clausulas axioma (reglas de la gramática). Aún más analizar una frase es una computación en un sistema de deducción automática. Si este sistema es el lenguaje Prolog se dispone de un analizador top-down eficiente para el lenguaje reconocido por la gramática.

Las gramáticas de clausulas contienen todas las caracteristicas que hacían importantes a las gramaticas de contexto libre: Las distintas formas de frases correctas de un lenguaje son descritas de forma clara y modular, permite representar en forma recursiva la estructura recursiva de frases (caracteristica muy comun) y existen buenos resultados del metodo de diseño de analizadores. Sin embargo las gramaticas de contexto libre no son adecuadas para la descripción de los lenguajes naturales. Las Gramáticas de clausulas (DCG de Definite clause grammars) las extienden en los aspectos siguientes:

- Permite expresar caracteristicas de dependencia del contexto.
- Permite construir diversos tipos de estructuras en la fase del analisis para las frases del lenguaje (la construcción no se ve restringida por la estructura recursiva de la gramatica).

. Permite incluir extra-condiciones en las reglas de la gramatica (de esta forma puede ser dirigido el analisis por diferentes metodos y razones) con argumentos en los simbolos no terminales.

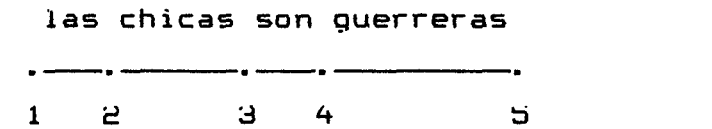
DEFINICION Y CARACTERISTICAS

Una gramática de clausulas es una cuaterna (V_n,V_t,R,S). Donde V_n es una extensión del V_n de la gramática de contexto libre, pues forman parte de este conjunto términos compuestos (predicados con argumentos como frase(x,y)). La parte derecha de una regla ß ($ß \in (V_n \cup V_t)^*$ en la gramática de contexto libre), pueden aparecer ademas llamadas Prolog.

Dada la gramática de contexto libre $GCL=(V_n,V_t,S,R)$, con $\alpha \rightarrow ß \in R$ sii $\alpha \in V_n$ y $ß \in (V_n \cup V_t)^*$ tal que $S \in V_n$ y $V_n \cap V_t = \emptyset$. Se construye la GCD correspodiente:

a. Para todo $x \in V_n$ se construye un predicado con dos argumentos $x(p_1,p_2)$ donde p_1 indica el punto de la frase donde comienza el constituyente denominado x, y p_2 indica donde acaba.
b. Para cada $x \in V_t$ se utiliza el predicado ternario conecta(x.y,x,y) , $x \in V_t$, $y \in V_t^*$, indicando que x conecta el punto x.y de la frase con el punto y de la frase.
c. Cada elemento de una regla es sustituido por su predicado correspondiente.

La frase se formaliza con una lista ordenada de aristas etiquetadas por simbolos terminales y con vértices etiquetados con la lista de simbolos terminales que etiqueta las aristas que parten de ese vertice y que llegan hasta el final. Ejm:



Donde los números representan a las listas de palabras: 1 =

las.chicas.son.querreras.nil, 2 = chicas.son.querreras.nil, 3 = son.querreras.nil, 4 = querreras.nil y 5=nil.

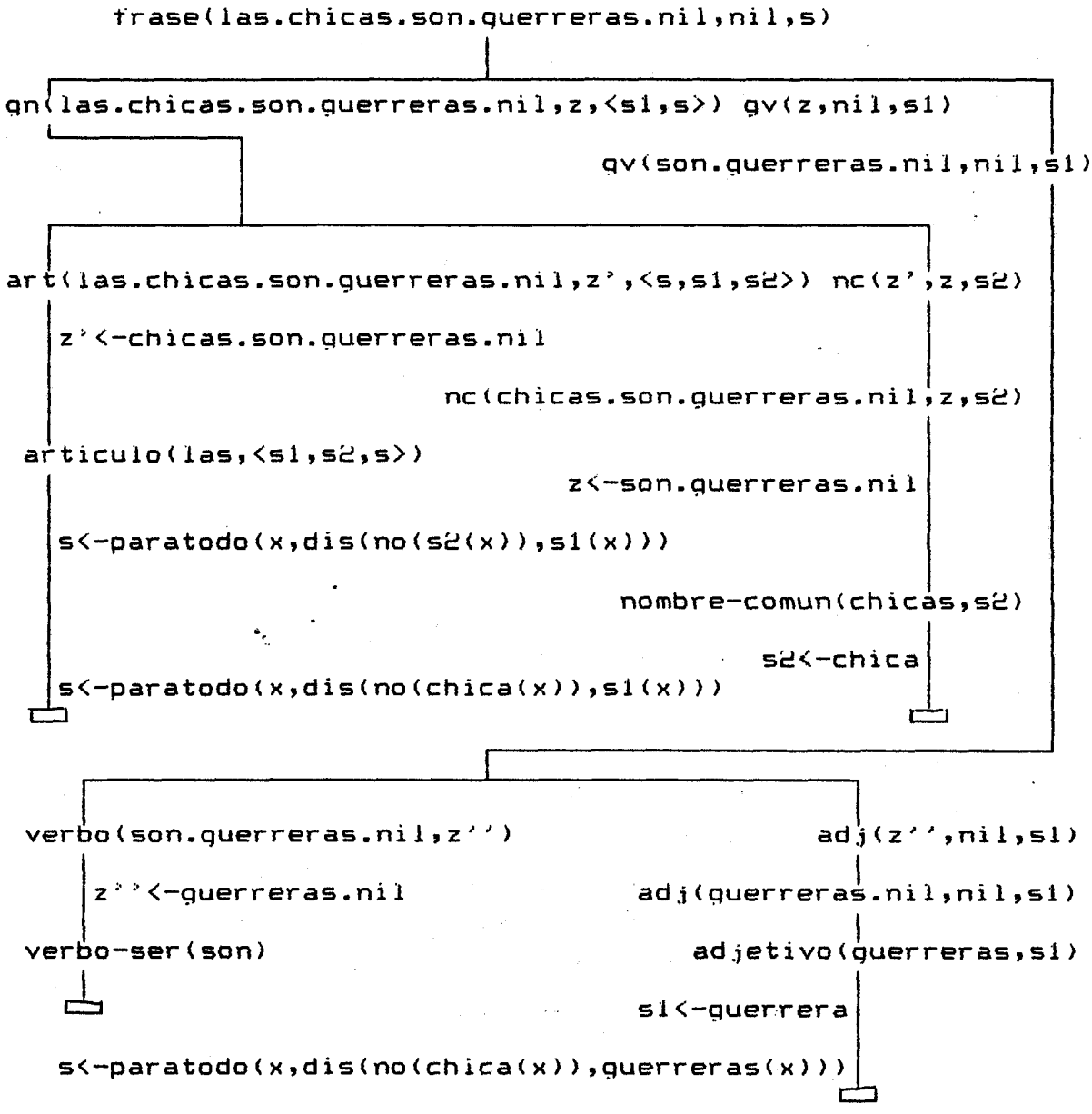
La construcción de estructuras (sintáctica o semántica) se realiza mediante la adición de argumentos a los predicados generados por elementos de V_n o V_t. De esta forma, la estructura (árbol de derivación por ejemplo) es construida por unificación durante la ejecución. Pueden ser expresadas restricciones a verificar por las instanciaciones de los argumentos de los predicados de una regla:
Fecha(x,M) -> mes(m), [x] (0<x<32).

También pueden ser utilizados los argumentos para transmitir información contextual:
gn(<p,q>,g) <- art(p,g) nc(q,g) (igual genero g en art y nc)

Cada tipo de ejecución de las cláusulas que forman una gramática corresponden a diferentes estrategias de análisis. Si son ejecutadas con la estrategia Prolog, entonces el análisis será top-down, de izquierda a derecha con backtracking en un punto de fallo al inmediato anterior donde haya opciones a seguir, distintas de la tomada. El analizador, por tanto, es tan eficiente como el Prolog.

Ejecutando las reglas aunque las estructuras pueden ser construídas paso a paso, de forma que en cada momento pueden tener partes aun no conocidas (representadas por variables) que han de serlo al final del análisis (proceso de unificación posterior), no necesariamente han de ir paralelas al proceso de construcción, como en el ejemplo:

```
frase(x,y,s) -> gn(x,z,<s1,s2>) gv(z,y,s1);

gn(x,y,<s1,s>) -> art(x,z,<s1,s2,s>) nc(z,y,s2);
gv(x,y,s) -> verbo(x,z) adj(z,y,s);
verbo(x.y,y) -> verbo-ser(x);

art(x.y,y,<s1,s2,s>) -> articulo(x,<s1,s2,s>);
articulo(la,<s1,s2,existe(x,conj(s2(x),s1(x)))>) ->;
articulo(las,<s1,s2,paratodo(x,dis(no(s2(x)),s1(x)))>)->;

nc(x.y,y,s) -> nombre-comun(x,s);
nombre-comun(chica,chica) ->;
nombre-comun(chicas,chica) ->;

adj(x.y,y,s) -> adjetivo(x,s);
adjetivo(querrera,querrera) ->;
adjetivo(querreras,querrera) ->;
```



Pereira y Warren [80] muestran como realizar la traducción de una red de transicion (ATN) a gramatica de contexto libre y como no siempre es posible el paso contrario. Basicamente una red de transición simple (sin ciclos y con únicos nodos inicial y terminal) puede ser pasada a DCG siguiendo los siguientes pasos:

a. La red simple corresponde a un no terminal.
b. Cada camino desde el nodo inicial al final se transforma en una regla para el no terminal asociado.
c. El cuerpo de cada regla es la traducción en orden de cada arco del camino.

Ademas muestran la potencia, eficiencia y claridad de los formalismos lógicos para las gramaticas y en particular de las gramaticas de clausulas, comparandolas con las ATN por ser estas últimas ampliamente utilizadas hasta ese momento. La estructura de su estudio sigue basicamente los puntos definidos por Woods [70] para expresar las ventajas de las ATN.

Perspicuidad: Cualidad de ser "fácil de entender" para minimizar el esfuerzo de construcción.

Una DCG, como una ATN, ha de ser entendida como una "maquina" que analice lenguajes. Una DCG además puede ser entendida como una descripción del lenguaje. Woods dijo (refiriéndose a las gramáticas de contexto libre): "observando una regla es posible comprender cuáles son los tipos de construcciones que permite." Esto no es tan claro (según Pereira y Warren) si no se hace una explicación o una buena elección de los elementos no terminales de la gramática (es decir, no es tan claro ver qué conjunto de partes de frases del lenguaje pueden ser construídas con esa regla).
Para explicar formalmente cómo una ATN define el lenguaje, es necesario utilizar la noción de cómo es ejecutada. Conceptualmente un ATN es que un mecanismo particular de análisis del lenguaje con estrategia top-down,

izquierda-derecha y con el orden expresado implícitamente en los registros con los que opera. Aunque podría haber partes explicables formalmente, siempre será difícil hacerlo para ciertos elementos, como por ejemplo la función de los registros (imposible de definir no utilizando términos de computar). Sin embargo, una DCG puede ser explicada formalmente en términos de la semántica declarativa de una cláusula. La semántica declarativa de una cláusula indica cuáles son los términos aceptados en el lenguaje, con lo que sería posible realizar una enumeración de todas las frases del lenguaje. En ningún momento será necesario utilizar el concepto de "ejecutar" cláusula.

Otra razón por la que se considera más clara a una DCG que a una ATN , es por la mayor modularidad de la primera y por la no existencia de variables globales (permitiendo el estudio independiente de las cláusulas). En las ATN el menor elemento es una subred, que en general es grande y complicada (como se observaal examinar la red de LUNAR).

Otra característica de una DCG es que una variable, cuando toma un valor, no puede modificarse y por lo tanto no se producen efectos secundarios. En las ATN sería posible modificar el contenido de los registros, pero en general los creadores de ATN no suelen utilizar esta facilidad para evitar los efectos secundarios.

Finalmente, el uso del pattern-matching, y no de los test explícitos o acciones de construcción, así como la posibilidad de utilizar solo un formalismo (las cláusulas) para describir e implementar en lugar de dos (las redes de transición para describir y el lenguaje LISP para implementar) aportan claridad al método.

Potencia y generalidad: La potencia de una ATN es equivalente a la de una DCG. Con respecto a la generalidad:

a. En las DCG es posible construir estructuras complejas que no son imagen de la recursividad propia de la gramática y

además, permite construir a un tiempo varias.

b. En el sentido de que ATN es una máquina particular de análisis, mientras que una DCG es en primer lugar un lenguaje de descripción y en segundo lugar puede ser sujeto a cualquier tipo de implementación.

c. Woods comentaba cómo sería factible con un ATN generar en lugar de reconocer frases simplemente. Pero los cambios que es preciso introducir son cuantiosos e incluso costosos. Sin embargo la DCG no necesita ninguna modificación a nivel de descripción, de formalismo ni de algoritmo.

d. La estrategia de análisis con una DCG no tiene por qué ser necesariamente la del Prolog.

Concisión: Se refiere a los puntos "eficiencia de la representación" y "tratamiento de regularidades" de Woods que pueden ser resumidos en El Principio de economización: la mejor gramática es aquella que describe un lenguaje con el menor número de símbolos.

Las gramáticas formalizadas con una DCG suelen ser la mitad en tamaño que las escritas con ATN debido fundamentalmente al uso del pattern-matching en lugar de operaciones explícitas de comparación y almacenamiento. Woods decía que las ATN permiten fundir las partes comunes de varias reglas de contexto libre, consiguiendo una representación más eficiente. Esto también puede ser realizado con cláusulas. En las ATN se realiza con el tratamiento especial de los registros, complicando de esta forma el nivel de descripción y la codificación en LISP. La modularidad de las DCG obligan al programador a construir partes separadas para partes conceptualmente diferentes de la gramática y no facilita la fusión de partes similares.

Eficiencia: La eficiencia operacional de un formalismo es fundamental para las aplicaciones y en el caso de las DCG, depende de la eficiencia del intérprete o compilador Prolog a utilizar. Las ATN necesitan un intérprete o compilador especial y siempre, al menos, una parte intermedia del de LISP. Los aspectos en los que implementaciones Prolog

compiladas se espera que sean màs eficientes que las actuales implementaciones compiladas de ATN son:

a. Compilar una DCG es un proceso a un solo nivel, sin necesidad de un lenguaje intermedio.

b. El acceso a valores de variables es inmediato.

c. La construcción de estructuras es realizada durante la ejecución del análisis sin costes externos.

d. La selección de las alternativas en las gramáticas es automática (propia del Prolog).

Con respecto a comparación de tiempos no es facil de realizar.

Flexibilidad: Un formalismo ha de ser flexible en el sentido de permitir su verificación, experimentación e introducción de nuevas ideas. Ambos formalismos lo permiten de diversas formas (las DCG no estan sujetas a ninguna necesidad de lenguajes intermedios).

Adecuación al trabajo teórico: Las DCG son adecuadas por su semántica declarativa (indicando "que") independientemente del mecanismo de ejecución. Aparte de que la lógica siempre será más idónea para expresar "que's" a cualquier tipo de usuario, que otras estructuras artificiales por muy gráficas o sencillas sean.

III.4 GRAMATICAS DE EXTRAPOSICION

Las gramáticas de "left extraposition" fueron diseñadas por F. Pereira para expresar claramente construcciones gramaticales en las que aparecieran referencias a objetos, en lugar de objetos explícitos subconstituyentes de frases (frases relativas o interrogativas). Captura las restricciones globales de la extraposición a izquierdas con el formalismo de los "corchetes" (o gap), describiendo los conceptos de forma independiente a representación en gramáticas de cláusulas para posteriormente ser interpretadas directamente por el lenguaje Prolog.

En términos gramaticales, hay "extraposición a la izquierda" en una frase cuando un subconstituyente de un constituyente (parte de la frase) es representado por otro a la izquierda del que está incompleto. Pereira llama "traza" al indicador que ocupa el hueco dejado por el objeto elidido y "marca" al objeto que lo representa. De esta forma puede decirse que el constituyente representado por la "traza" ha sido extrapuesto a la izquierda y en esta nueva posición es representado por la marca.
Las frases relativas son tratadas de esta forma, siendo el pronombre relativo la "marca" y la frase que le sigue estará formada por una traza y otros subconstituyentes:

El hombre que $[t_i$ viene $]$ es un profesor
m_i frase

Hay una restricción: un pronombre relativo que aparece fuera de un grupo nominal, no puede estar acotado por una "traza" que aparezca dentro de una frase relativa subconstituyente del grupo nominal. Es decir, no es posible la configuración:

$m_1 \ldots [\ldots [\ldots m_2 [\ldots t_2 \ldots t_1 \ldots]] \ldots]$
gn frel frase

Ejemplo:

Sea la gramatica de contexto libre que trata frases relativas:

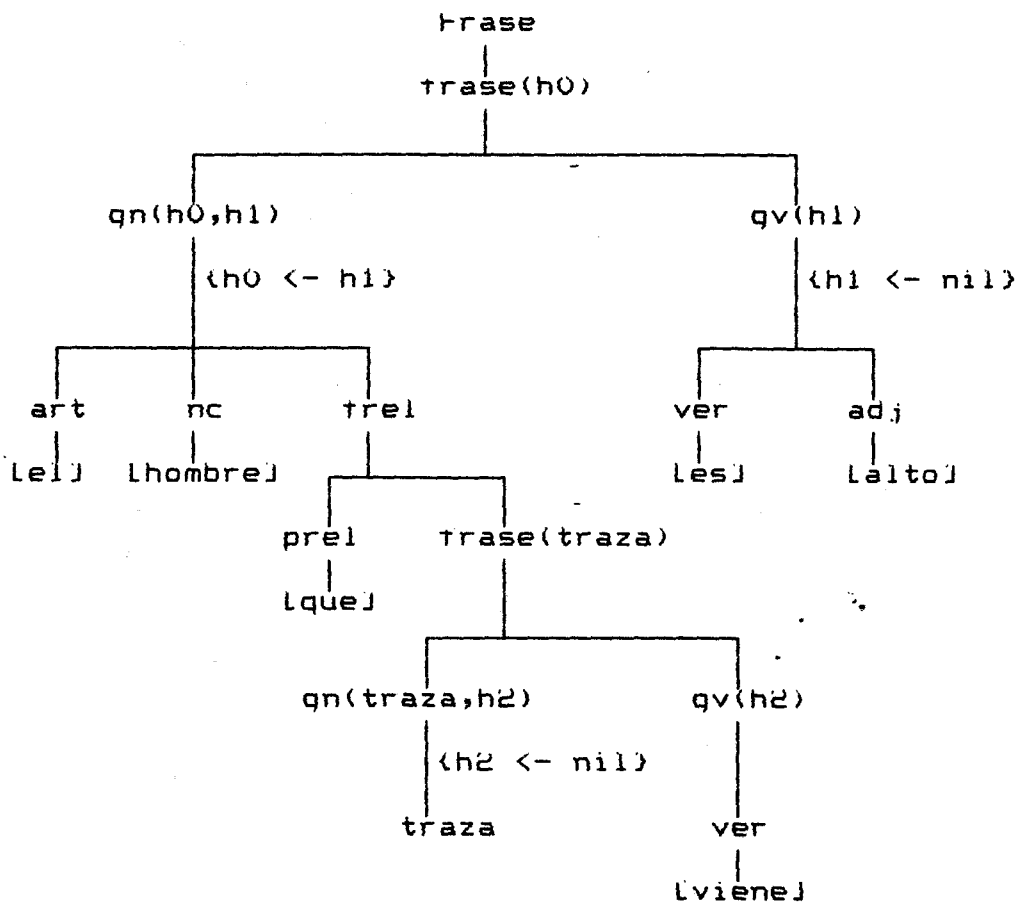
```
frase -> gn gv
gn -> np / art nc frel / art nc gp / traza
traza -> []
gv -> ver / ver gn / ver adj
frel -> [] / prel frase
gp -> prep gn
```

Modificada para constituir una gramatica de contexto libre que trata frases relativas con la idea de extraposición:

```
frase -> frase(h0)
frase(h0) -> gn(h0,h1) gv(h1)
gn(h,h) -> np / art nc frel
gn(h0,h1) -> art nc gp(ho,h1)
gn(traza,nil) -> traza
traza -> []
gv(nil) -> ver / ver adj
gv(h) -> ver gn(h,nil)
frel -> [] / prel frase(traza)
gp(h0,h1) -> prep gn(h0,h1)
```

Los argumentos variables "hi" tomaran el valor "traza" si hay a la derecha algun gn extrapuesto o "nil" si no.

Ejemplo de análisis:


frase
frase(h0)
qn(h0,h1)
qv(h1)
(h0 <- h1)
(h1 <- nil)
art
nc
frel
ver
adj
[el]
[hombre]
[les]
[alto]
prel
frase(traza)
[que]
qn(traza,h2)
qv(h2)
(h2 <- nil)
traza
ver
[viene]

Sin embargo esta manera de hacer puede resultar complicada en la acción y poco legible la gramática que la describa. Las gramáticas de extraposición intentan ser la solución a este problema: que sea facil la expresión de las relaciones entre marcas y trazas y que la descripción de la gramatica sea clara.

Para ello en la parte izda de las reglas de la gramática se permite que aparezcan varios simbolos (en lugar de uno solo).

frel -> []

frel -> mrel frase

mrel ... traza -> prel

la última regla indica que un pronombre relativo es analizado si es posible encontrar una secuencia formada por una marca "mrel", un conjunto indeterminado de constituyentes "..." y una traza.

DEFINICION

Una regla de esta gramática es de la forma

$s_1 \ldots s_2$ etc $s_{k-1} \ldots s_k$ -> r

donde s_1 es un simbolo no terminal, s_2, s_{k-1} y s_k son simbolos terminales o no terminales, r la parte derecha de la regla analoga a la parte derecha de las reglas de las gramáticas de clausulas definidas y "..." secuencias no especificadas de simbolos terminales o no terminales. Estas reglas pueden ser entendidas como:

$s_1 \ldots s_2$ etc $s_{k-1} \ldots s_k$ -> r ... ...

(a) (b) (a) (b)

donde:

(a) < ... > es una cadena entre corchetes "balanceada"

(b) $s_1 \ldots s_2$ etc $s_{k-1} \ldots s_k$ -> r regla de extraposición puede ser aplicada a la cadena: s = x_0 s_1 x_1 s_2 etc x_{k-1} s_k x_n si cada "hueco" x_i es balanceado.

El resultado de la aplicación es la cadena:

x_0 r <<..< x_1 > x_2 >..x_{n-1} > x_n. Ejm:

Regla: mrel ... traza -> prel, Cadena: mrel Juan ama traza, Produce: prel <Juan ama>

Cuando se aplican las reglas hay que tener en cuenta que interaccionan entre ellas. Ademas despues de la aplicación de este tipo de regla no se pueden aplicar otras que tomen elementos de dentro y de fuera de la secuencia entre corchetes. Para no violar la restricción utilizando el formalismo de los corchetes balanceados, hay que cambiar la regla de frel de la siguiente forma:

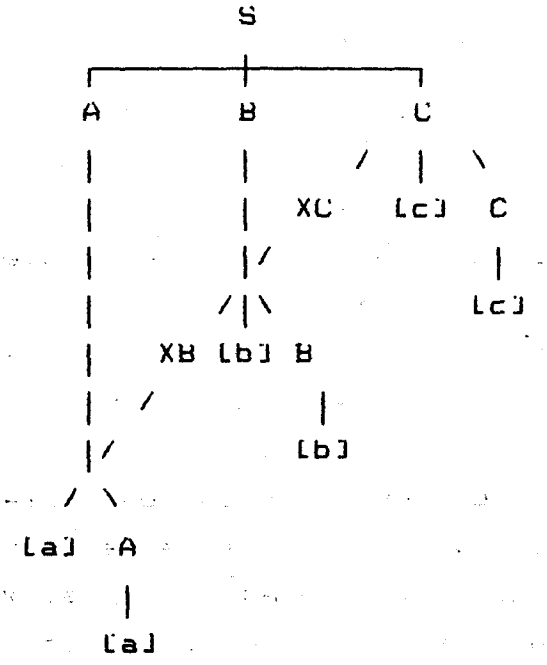
frel -> op mrel frase cl

op ... cl -> []

Una frase analizada correctamente por una gramatica de extraposición a la izquierda coincide con la frase obtenida por aplicación de reglas a partir del simbolo inicial al serle eliminados todos los corchetes. En los arboles de derivación, representación de la estructura de la frase, puede haber nodos que comparten aristas.

Ejemplo: Gramática de extraposición para $\{a^n b^n c^n \mid n>=1\}$

{S->ABC; A->[a]; A...XB->[a] A; B ->[b]; B...XC->XB [b] B; C->[c]; C->XC [c] C}



La implementación de estas gramáticas es un programa que las traduce a clausulas de Horn, para ser ejecutadas con Prolog.

III.5 GRAMATICAS DE ESTRUCTURA MODIFICADA

Las Gramáticas de estructura modificada [Dahl McCord 83] son una extensión de las de Extraposición en el aspecto siguiente. En una gramática de extraposición son los argumentos de los predicados de las reglas los que restringen su utilización y contienen la estructura a asociar a las frases correctas. En una gramática de estructura modificada se realiza la segunda tarea en forma implicita.

Las reglas son de la forma:

A:Sem -> B.

A ->B

donde A->B es una regla de extraposición y Sem es un término que forma parte de la semántica de la frase si la regla es aplicada. Sem es de la forma: operador-formalógica donde operador indica la forma en que afecta este elemento en la componente semántica construida hasta ese momento y formalógica es lo que aporta la regla a la estructura semántica. En el segundo tipo de regla puede considerarse que Sem es la estructura trivial vacia (1-true).

Un ejemplo de este tipo de gramáticas es:

```
frase -> gn(x) gv(x);
gn(x) -> gp(x) nc(x);
gn(x) -> np(x);
gv(x) -> ver(x,y) gn(y);
gp(x) -> prep(x);
gp(x) -> art(x);
prep(x):Sem -> [p] {det(p,x,Sem)};
art(x):Sem -> [a] {articulo(a,x,Sem)};
nc(x):1-pred -> [n] {nombre-comun(n,x,pred)};
np(n) -> [n] {nombre-propio(n)};
ver(x,y):1-pred -> [v] {verbo(v,x,y,pred)};

"léxico"
nombre-comun(hombre,x,hombre(x)) ->;
```

```
nombre-comun(mujer,x,mujer(x)) ->;
nombre-propio(Manuel) ->;
nombre-propio(Alicia) ->;
verbo(ve,x,y,ve(x,y)) ->;
verbo(ama,x,y,ama(x,y)) ->;
det(cada,x,paratodo(x,y,z)) ->;
det(a,x,existe(x,y,z)) ->;
articulo(un,x,existe(x,y,z)) ->;
articulo(una,x,existe(x,y,z)) ->;
```

Los tests de las reglas entre llaves son objetivos Prolog (en el ejemplo son llamadas al léxico). Esta gramática asocia estructura semantica a las frases que reconoce:

Juan ve a Maria con semántica ve(juan,Maria);

Cada hombre ama una mujer
paratodo(x,hombre(x),existe(y,mujer(y),ama(x,y))

La estructura sintáctica es un árbol llamado "estructura modificada" que si se aplican reglas del primer tipo es de la forma: sin(Nt,Sem,Mod) siendo Nt el simbolo de predicado que aparece en A y Mod la lista de las estructuras asociadas a las subfrases analizadas con la parte derecha de la regla. Se considera un arbol con un nodo padre etiquetado por Nt y Sem y con un nodo hijo etiquetado por Mod. En el ejemplo anterior para la frase "Cada hombre ama una mujer" la estructura sintáctica es:

```
<frase,l-true>
  <gn(x),l-true>
    <gp(x),l-true>
      <prep(x),paratodo(x,y,z)>
    <nc(x),l-true>
  <gv(x),l-true>
    <ver(x,y),l-ama(x,y)
    <gn(y),l-true>
```

<gp(y),l-true>

<art(y),existe(y,z,u)>

<nc(y),l-mujer(y)>

La implementación realizada en Prolog-10 es un interprete que dirige el analisis sintáctico y la interpretación semántica. La estrucutra asociada a las frases es un arbol con nodos etiquetados por elementos sintácticos (habituales) y elementos semánticos que presentan la particularidad de que aportan información en forma aislada y no dependiente de los nodos sucesores, siendo estos los que pueden actuar sobre esta estructura "modificandola". La interpretación semántica se realiza en dos pasos, en el primero se estudian las posibles diferencias entre las relaciones sintácticas y las semánticas y la segunda realiza las modificaciones referidas anteriormente.

Estas gramáticas permiten tratar la extraposición y la coordinación (fueron creadas para ello). El interprete maneja la coordinación por si sólo sin necesitar que aparezcan elementos de ella en la gramática pero reflejando en la parte semántica las caracteristicas pertinentes.

El sistema puede considerarse eficiente por varias razones. Una de ellas es que se ha implementado mediante programación lógica compilable. Otra es que el interprete no es muy grande y además no guarda toda la historia de cada ejecución.

III.6 GRAMATICAS GAPPING

Son gramáticas lógicas que permiten trabajar en forma explicita con huecos o gap's (subcadenas no identificadas como constituyentes) entre elementos de frases. Con ello es posible realizar un tratamiento en cualquier orden de las palabras de una frase. Aparece el simbolo lógico "gap(x)" en el vocabulario no terminal de la gramática y es entendido como una subcadena de la frase a analizar que es separada, colocada en otra parte de la frase y no analizada hasta que se llegue a la nueva posición.

Estas gramáticas son una generalización de las gramáticas de extraposición ya que los "gap" pueden aparecer en cualquier posición de la parte derecha de la regla.

DESCRIPCION

Utilizan un símbolo no terminal gap(G) para referirse a una subcadena G, composición de constituyentes gramáticales que serán analizados posteriormente.
Una regla en esta gramática es de la forma :

$$A,\ \alpha \rightarrow \beta$$

donde $A \in V_n - \{gap(G)\}$, $\alpha, \beta \in (V_n \cup V_t \cup \{gap(g)$ U llamada Prolog$\}^+$.
(La notación introduce entre [] a simbolos terminales, y entre {} a llamadas Prolog).

Sea F un conjunto de símbolos funcionales más el símbolo unario "gap", V un conjunto enumerable de variables, R un conjunto de términos construido a partir de F y V.
$x \in R$ sii:

(1) $x \in V$

o (2) $x \in \{f \in F \mid aridad(f)=0\}$

o (3) $x = f(t_1 \ldots t_n)$

donde $f \in \{f \in F \mid aridad(f)=n\}$ y t1...tn son términos.

Sean H conjunto de los términos de base (sin variables)

de R, $V_t \subset R$ el vocabulario terminal, $V_n \subset R$ el vocabulario no terminal ($V_t \cap V_n = \emptyset$), $\Gamma \subset V_n$ el conjunto de símbolos "gap", $\Gamma = \{gap(G_1),...gap(G_n)\}$ con G_i variables, $V_s \subset V_n$ el conjunto de símbolos iniciales.

P es el conjunto de reglas de la forma:

$$nt\ ,\alpha_0,\ gap(G_1),\ \alpha_1,...,gap(G_n),\ \alpha_n \rightarrow \beta_0,\ gap(G_{i1}),\ \beta_1,\ ...,gap(G_{im}),\ \beta_m$$

donde $m,n>=0$, $nt \in V_n$, $\alpha_i,\beta_j \in (V_n \cup V_t)^*$, $gap(G_i) \in \Gamma$ y $1<=i,j<=n$.

Se define => relación de reescritura sobre $(V_n \cup V_t)^*$:

$u \Rightarrow v$

sii existe una regla $a_0, gap(G_1),....a_n \rightarrow \beta_1,gap(G_{i1}), ...\beta_n$ y existe una sustitución Φ de términos por variables tal que:

$$u\Phi = (a_0,\tau_1 a_1 ... \tau_m a_m)\Phi$$

para algun $\tau_i \in (V_n \cup V_t)^*$ y $v=(\beta_0\ \tau_{i1}\ \beta_1 ... \tau_{im}\ \beta_m)\Phi$.

A $G=(V_n,V_t,\Gamma,V_s,P)$ se denomina "gapping grammar" siendo L(G) el lenguaje definido por G:

$$L(G)= \{\ t \in Vt^* \mid s \in Vs,\ s \Rightarrow t\ \}$$

No se han incluido llamadas Prolog en la definición formal de la gramática. Si aparecen en una regla son afectadas por la sustitución al ser aplicada la regla y se ejecutan cuando se aplica la regla. Su fallo produce el fallo de aplicación de la regla. Las gramáticas son escritas (en la práctica) a partir de términos de base $t \in V_t^* \cap H$.

Ejemplo:

frase(and(s_1,s_2)) -> frase-af(s_1), conj, frase-af(s2).

frase-af(s) -> nom-p(k), verbo(k,p,s), objeto(p).

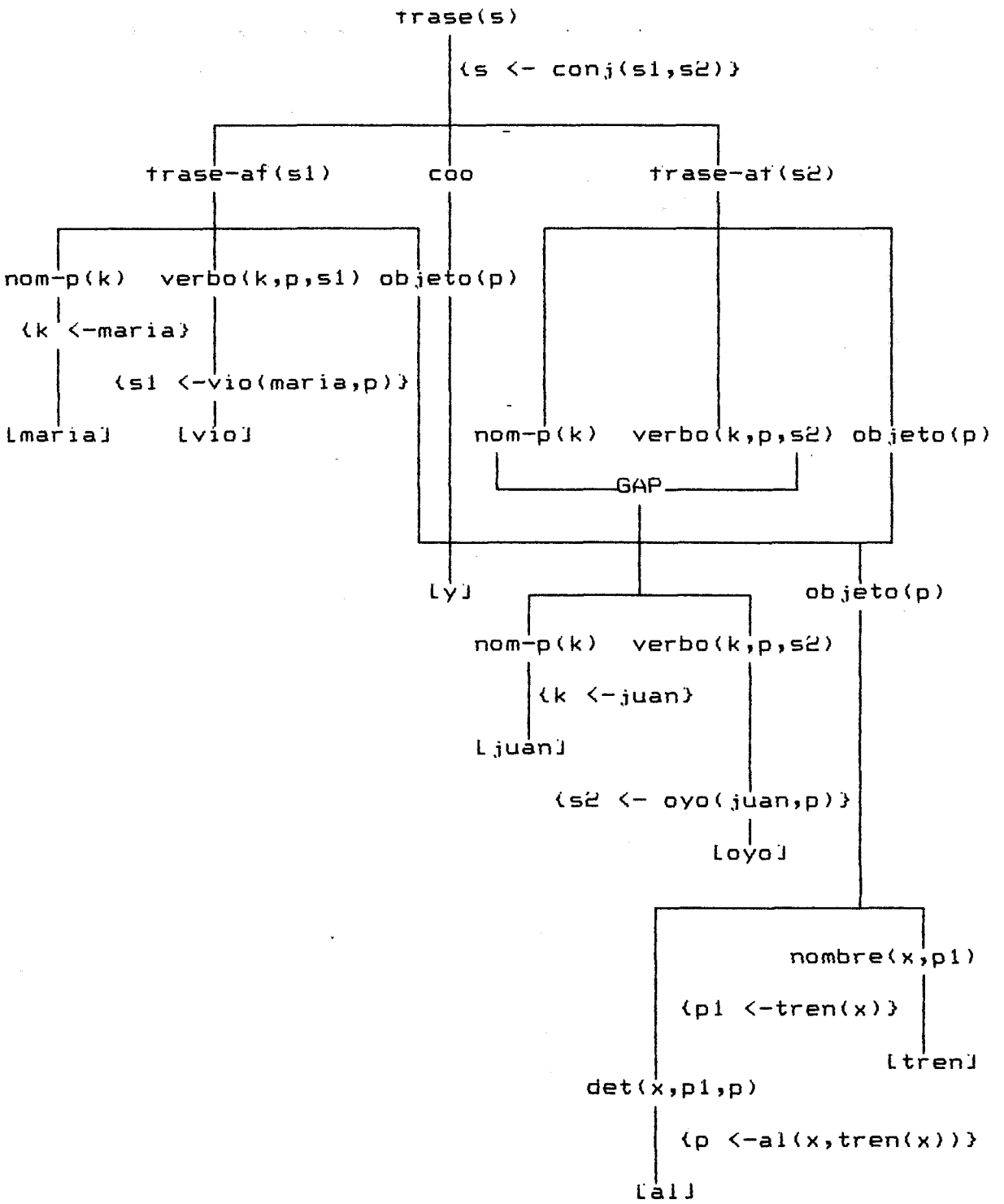
```
objeto(p) -> det(x,p1,p), nombre(x,p1).
objeto(p) -> adj(p).
objeto(p), conj, gap(G), objeto(p) -> [y], gap(G), objeto(p).

det(x,p,al(x,p)) -> [al].

nombre(x,tren(x)) -> [tren].

nom-p(maria) -> [maria].
nom-p(juan) -> [juan].

verbo(x,y,ve(x,y)) -> [ve].
verbo(x,y,oye(x,y)) -> [oye].
```

La tercera regla de "objeto" permite tratar la elipsis de un "objeto" en la primera frase, reconstruyéndolo por unificación del "objeto" en la segunda frase.

Ejemplo de analisis de la frase:
"maria vio y juan oyo al tren"

con semántica
conj(vio(maria,al(x,tren(x)),oyo(juan, al(x,tren(x)))

y estructura sintáctica:


frase(s)
{s <- conj(s1,s2)}
frase-af(s1)
coo
frase-af(s2)
nom-p(k)
verbo(k,p,s1)
objeto(p)
{k <-maria}
{s1 <-vio(maria,p)}
[maria]
[vio]
nom-p(k)
verbo(k,p,s2)
objeto(p)
GAP
[y]
objeto(p)
nom-p(k)
verbo(k,p,s2)
{k <-juan}
[juan]
{s2 <- oyo(juan,p)}
[oyo]
nombre(x,p1)
{p1 <-tren(x)}
[tren]
det(x,p1,p)
{p <-al(x,tren(x))}
[al]

Las gramáticas de metamorfosis están incluídas en las gramáticas gapping (GG) ya que para cada símbolo "nt" de la parte izquierda de la regla GG necesitamos en las de metamorfosis una regla de la forma nt -> [nt] donde [nt] era un pseudo-terminal que no aparece en la frase de entrada.

Las gramáticas de extraposición son un caso especial de las gramáticas GG, donde todos los gaps de la parte izquierda son reescritos en orden secuencial al final de la parte derecha de la regla. Además las gramáticas de extraposición no permiten el uso de llamadas Prolog en la parte izquierda de la regla. Finalmente cabe destacar que las gramáticas GG no necesitan utilizar marcas para la continuación del análisis y son mas eficientes sin perder poder expresivo como muestra el siguiente ejemplo:

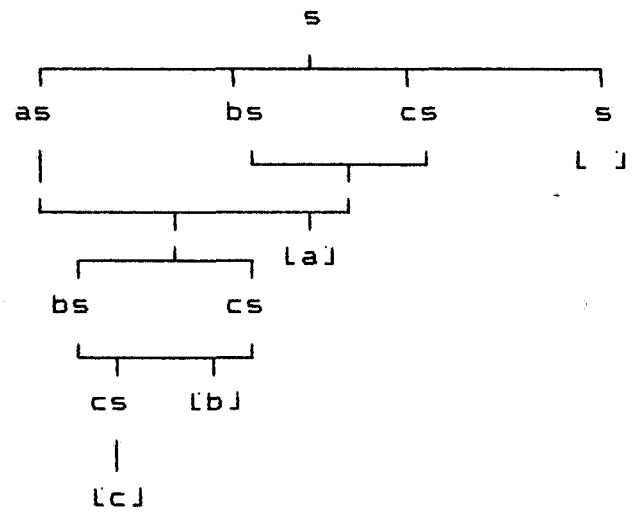
"scrambled problem"
L(G) = {$a^n b^n c^n$ en cualquier orden}

Gramática de Extraposición:
s -> as, bs, cs, s
s -> []
xs -> [x]
xs,gap(G),x' -> [x'],xs,gap(G) con x∈{a,b,c},x'∈{a,b,c}-{x}
luego son necesarias dos reglas para el comienzo y 9 reglas más para generar o analizar las palabras; y son (3)**3 + 2 reglas (3=número de letras).

Gramática gapping:
s -> com, s1, fin.
s1 -> as, bs, cs, s1.
s1 -> [].
com, gap(g), as -> [a], com, gap(G).
com, gap(G), bs -> [b], com, gap(G).
com, gap(G), cs -> [c], com, gap(G).
com, fin -> []
luego sólo se necesitan m reglas (m=número de letras) más 4.

Por lo tanto el análisis será más eficiente y sin embargo la gramática sigue explicando claramente el proceso.
Todavía la gramática GG anterior puede ser más concisa :

s -> as, bs, cs
s -> [ ]
xs, gap(G) -> gap(G), [x] x∈{a,b,c}



Este ejemplo muestra un tratamiento conciso, claro y eficiente del análisis de lenguajes sin orden fijo en las palabras que componen las frases. Caracteristica importante ya que la posición de las palabras en las frases en general indica énfasis u otras peculiaridades y es practica aceptada en la mayor parte de los lenguajes naturales.

Las gramáticas GG posibilitan mezclar palabras, en distinto orden, de distintos constituyentes de las frases (como en las leyes o anuncios). Es posible definir gramáticas con ordenes generales establecidos entre los componentes como en el ejemplo siguiente:

(orden (DC) y regla A->B,C,D)

se representa al conjunto:

(A ->DBC, A ->DCB, A ->BDC)

Pero estas gramáticas comienzan a ser complejas con la complejidad del orden establecido.

Otro problema que estas gramáticas pueden subsanar es el tratamiento del orden de los constituyentes que deben o no aparecer según lo determinen otros constituyentes. En el ejemplo que se presenta a continuación se muestra además como por ser variables los argumentos de los "gap" son posibles todas instanciaciones (incluyendo la vacia).

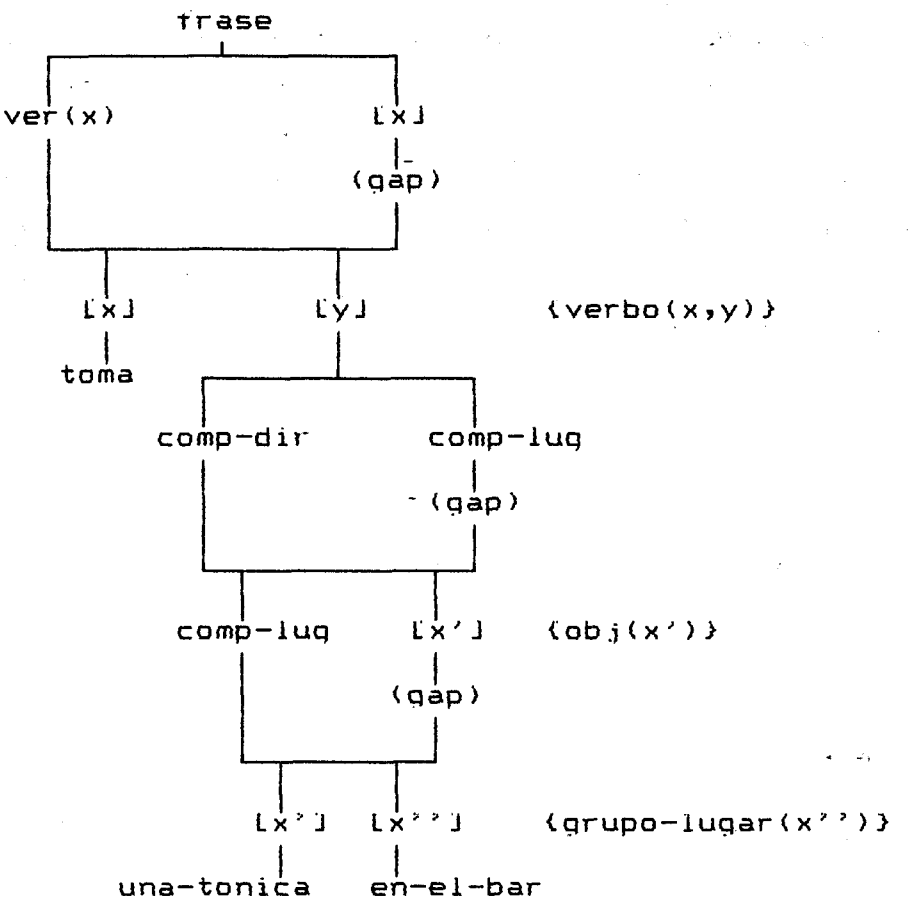
La dificultad de estas gramáticas está en la ejecución o proceso de los gaps: ¿qué conjunto de constituyentes se seleccionan para formar un gap?.

Es implementable tanto la extraposición a izquierdas como a derechas [Dahl, 84].

Ejem:

```
frase -> ver(x), [x].
ver(x), gap(g) -> gap(g), [y], {verbo(x,y)}
verbo(toma,{comp-dir,comp-lug}) ->;
comp-lug, gap(g) -> gap(g), [x], {grupo-lugar(x)}
comp-dir, gap(g) -> gap(g), [x], {obj(x)}
obj(una-tonica) ->;
grupo-lugar(en-el-bar) ->;
```

Un análisis realizado con esta gramática de la frase "toma una tónica en el bar" da lugar a la siguiente estructura:

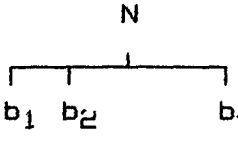

frase
ver(x)
[x]
(gap)
[x]
[y]
{verbo(x,y)}
toma
comp-dir
comp-lug
(gap)
comp-lug
[x']
{obj(x')}
(gap)
[x']
[x'']
{grupo-lugar(x'')}
una-tonica
en-el-bar

III.7 GRAMATICAS DE PUZZLES

Las gramaticas de puzzles [Sabatier, 84] intentan ser (no estan completamente definidas aun) un formalismo logico declarativo claro, simple y potente para describir gramáticas no triviales por parte de linguistas (que no tienen porque comprender los métodos procedurales). Con estas gramáticas el lenguaje se define en términos de arboles-key o piezas del puzzle representando a las reglas. Nuevos arboles o elementos del puzzle pueden ser construidos por "ensamblaje" de los ya conocidos. Las restricciones se asocian a los arboles-key y son evaluadas cuando es necesario (y tienen a sus argumentos con valor). Para realizar el analisis/generación de frases las reglas que definen la estrategia, dirigen el proceso de ensamblar elementos del puzzle.

DESCRIPCION

Un arbol-key es un arbol:



donde N es un atomo o predicado (no variable) y cada b_i es un simbolo no terminal, terminal, el simbolo nil o bien un arbol-key ya conocido. Dos arboles-key k y k' se ensamblan si siendo s_1 hoja de k y s_2 raiz de k' entonces s_1 y s_2 son unificables.

Una gramatica puzzle es un conjunto de pares:

$G = \{(k_1, c_1), \ldots, (k_n, c_n)\}$ (con $n>0$)

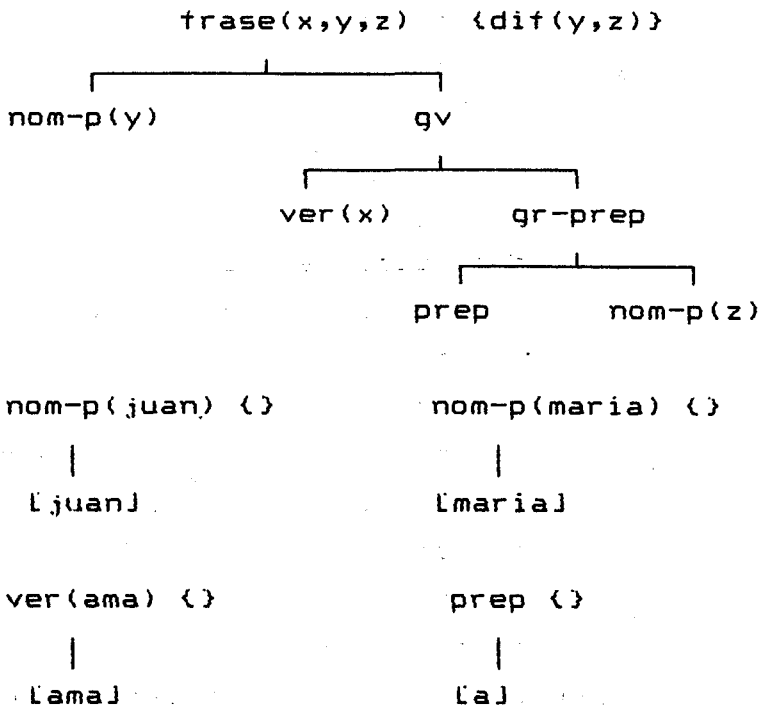
donde cada k_i es un arbol-key basico y cada c_i es una conjunción o disyunción de restricciones sobre k_i. Al par (k_i, c_i) se denomina pieza del puzzle.

Una cadena pertenece al lenguaje generado por una gramática puzzle si existe un arbol-key k, cuya lista de

hojas sea la cadena.

Como en otras gramaticas lógicas, los argumentos de los predicados pueden ser usados para especificar y construir las representaciones asociadas a la frase analizada o generada. Las condiciones asociadas a los arboles-key expresan restricciones en los valores de las variables argumento que aparecen en algun nodo. Son evaluadas tan pronto como son necesarias y siempre que se vonozca el valor de sus argumentos (los linguistas al dar las restricciones no se preocupan de como ni cuando será efectuada la computación).

Estas gramáticas pueden ser traducidas a gramáticas de clausulas y por ello interpretadas en Prolog. A pesar de esto su creador esta intentando definir estrategias que optimicen su utilizacion. Ejem:



Ensamblando estos arboles puede ser analizada o generada la frase "Juan ama a maria".

III.7 REALIZACIONES

Todos los formalismos de gramaticas lógicas expuestos en el apartado anterior han sido implementados en Prolog (ya que la operación de unificación y la estrategia en profundidad no determinista son operaciones básicas del lenguaje) y han dado lugar a aplicaciones de interfaces en lenguaje natural para bases de datos o sistemas expertos.
Se enumera a continuación algunas de estas realizaciones:

Banco de datos administrativo [Dahl 77]
V. Dahl durante su estancia en el GIA de Marsella construyó un sistema de interrogación de bases de datos en castellano utilizando una gramática de metamorfosis. La base de datos esta descrita con predicados y clausulas de Horn en lógica de primer orden, con lo que es posible acceder tanto a la información explicita de la base como a la deducible. Con objeto de tratar la negación por defecto (una formula es falsa si no es posible demostrar su verdad) trabaja sólo con dominios finitos. La parte linguistica del sistema esta separada del tratamiento de la base.

ORBI [Oliveira, Pereira et Sabatier 82]
Sistema experto en "enviromental biophisical resource evaluation" con un interfaz en lenguaje natural (portugues) capaz de explicar la aplicación de su concimiento a un territorio particular asi como de responder preguntas sobre sus capacidades linguistas o indicar porque acepta frases como correctas o incorrectas (ambas cosas en lenguaje natural o con menus). Ha sido implementado en Prolog y la gramatica de clausulas permite tratar la elipsis y la coordinación de frases. La interpretación de las frases analizadas es una lista optimizada de objetivos Prolog.
El sistema fué el primer producto terminado realizado totalmente en Prolog sobre un PDP 11 / 02 que consta de una base de hechos, una base de reglas de inferencia (capaz de responder preguntas rezonando sobre los hechos) y un

interface para realizar preguntas en portuques o a traves de menus.

Además por ser un sistema experto dispone de un módulo que testea un cierto tipo de coherencia semantica y produce las formulas asociadas a preguntas correctas. A partir de la formula se responde a la pregunta.

Dialogos en frances [Pasero 82]

Sistema de dialogo entre usuario y maquina que consta de un módulo de analisis sintactico-semantico basado en la lógica trivaluada (pero transformada a la bivaluada para su tratamiento) mas un módulo que hace posible la asimilación del conocimiento por la máquina.

Para cada frase declarativa (unica estructura de frase que se trata) el sistema extrae una información semantica y antes de añadirla a su conocimiento estudia que no sea contradictoria con la ya existente. Las presuposiciones asociadas a la frase no deben contradecir las informaciones anteriores. Si lo hace entonces la frase es considerada como absurda y no aporta nueva información al sistema. Las aserciones asociadas a la frase no pueden contradecir la información del sistema ya que si esto ocurre entonces la frase es considerada como falsa pero el sistema toma la información de la presuposición. Ejemplo de dialogo:

Us : Pedro vive en Marsella.
Sis: OK
Us : Pedro no vive en Marsella.
Sis: Falso
Us : Ninguna persona vive en Marsella.
Sis: OK
Us : Pedro es una persona.
Sis: Falso
Us : Las personas que viven en Marsella pesan 30 Kg
Sis: Absurdo

MICROSIAL [Pique, Sabatier 82]

Es un interface en frances para una base de datos de una administración militar. La gramática es de metamorfosis.

CHAT-80 [Pereira 83]

Es un programa escrito en Prolog que responde preguntas en ingles a una base de datos de geografia (no es un producto acabado). Dispone de un módulo de interpretación de la semantica de las frases. La gramática es de extraposición.

ORBIS [Colmerauer 83]

Sistema que se comunica con el usuario en ingles o frances en el dominio de la astrologia (planetas). Es una gramatica lógica, la representación del conocimiento se realiza con clausulas y esta implementado en Prolog.

[Pentonnen, M. 85]

Programa que permite preguntar en finlandes una base de datos escrita en Prolog.

CAPITULO IV:
VALORACION DE LA SITUACION ACTUAL Y PRESENTACION DEL SISTEMA SIRENA

IV VALORACION DE LA SITUACION ACTUAL Y PRESENTACION DEL SISTEMA SIRENA

Una vez realizado el resumen de las técnicas clásicas de análisis o comprensión del lenguaje natural en las dos líneas existentes (capítulos dos y tres) se pasa a continuación a hacer una valoración de la situación actual de este tipo de sistemas.

Se destacan los siguientes puntos sobre la situación actual de los sistemas de comunicación en lenguaje natural:

- Hay un "fracaso" de los sistemas estructurados clásicos y se ha desechado la idea de estructurar y realizar el estudio en fases independientes salvo en dominios muy concretos y reducidos.

Se crean nuevos paradigmas resultado de integrar los distintos aspectos en una u otra forma. Como eje de la implementación suele tomarse una gramática complementada con criterios procedentes de otros niveles. La orientación a seguir es una amalgama de aspectos léxicos, sintácticos, semánticos y pragmáticos mediante una base de conocimiento. Para ello se utilizan fundamentalmente técnicas del área de la Inteligencia Artificial pero no exclusivamente.

- Dentro de los sistemas "no muy estructurados" aparece una línea peculiar como es el formalismo de las Gramáticas Lógicas que permite formalizar y realizar conjuntamente las diferentes fases clásicas de análisis.

Esta técnica presenta la ventaja de ser aplicable a modelos integrados de análisis comprensión y resolución de problemas.

- No existe aún una única teoría asumida como adecuada para representar el significado de las frases.

Hay dos opciones, utilizar como base la lógica (teoría bien fundamentada) o bien utilizar modelos construidos al efecto.

- La mayor parte de los sistemas funcionan en forma efectiva para dominios únicos. En general un cambio de dominio exige una gran reconversión e incluso una nueva definición de los (o algunos) elementos del sistema de comprensión del lenguaje

natural. Claramente cualquier cambio de este tipo produce variaciones a tener en cuenta pero se ha de intentar que esta tarea sea lo mas "gramatical" posible, es decir que afecten al diccionario y en menor medida a la gramática.

- Aunque no es posible construir de momento un sistema que abarque un lenguaje natural en su totalidad (problemas graves con el diccionario o con la representación de estructuras linguistas), la elección del subconjunto de frases que se considera mas adecuado para la aplicación que lo utiliza es una tarea fundamental y no existe aún una delimitación asumida como válida por la mayoria.

Esta claro que este subconjunto ha de poder ser utilizado por distinto tipo de personas y ha de aportar los mecanismos para la expresión concisa y precisa de acciones a realizar.

Todo ello lleva a que cualquier interfaz que se desee utilizar en forma eficiente y agradable ha de ser lo mas amplio posible.

En línea con el apartado anterior y como propuesta de esta tesis se ha planteado un sistema de resolución de tareas en lenguaje natural que utiliza un formalismo y un tipo de proceso basando en la lógica. Este sistema se ha llamado Sirena (como ya se ha indicado en la introducción de esta tesis) y es un sistema completo de comprensión y comunicación que permite tareas de adquisición, representación, inferencia y explicación del conocimiento en castellano.

En un sistema de las caracteristicas del presentado aunque la formulación es elegante, cuando se pretende construir un sistema de ámbito muy amplio (dominio del discurso) se puede llegar, como otros sistemas, a una inviabilidad de tratamiento. Sin embargo para interfaces con dominios limitados puede construirse un sistema muy poderoso y a la par integrarse con sistemas de resolución de problemas creandose una estructura mixta de comprensión y procesamiento que se acercaria mucho al concepto de programación en lenguaje natural.

Las caracteristicas mas destacables del sistema Sirena son las siquientes:

1. Integra las diferentes tareas a realizar con un sistema basado en el conocimiento: procesos de adquisición y recuperación del conocimiento, de comprensión y resolución o de explicación de decisiones. Todo ello en castellano.
3. Dispone de un subconjunto adecuado y suficiente del castellano para la comunicación.
3. La adquisición de palabras a introducir en el diccionario del sistema se realiza interactivamente (no exige una previa definición).
4. Implementación óptima de un algoritmo de análisis del lenguaje que retrasa la verificación de restricciones hasta el momento adecuado permitiendo la recuperación de ciertos errores gramaticales pero resolubles.
5. El sistema Sirena se utiliza con diferentes aplicaciones basadas en el conocimiento y construidas con paradigmas similares.
Puede ser considerado un generador de interfaces de comunicación que dispone de una gramática cuyo lenguaje generado es un subconjunto del castellano y de un diccionario básico.

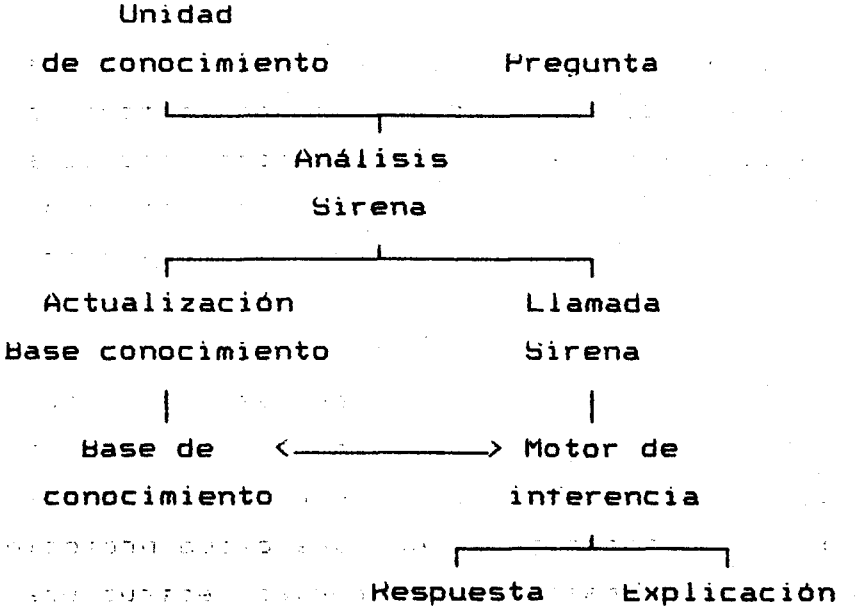
En el capítulo siguiente se realiza un descripción detallada del sistema Sirena.

CAPITULO V:
SISTEMA SIRENA

V SISTEMA DE PROGRAMACION EN LENGUAJE NATURAL: SIRENA

El objetivo de este trabajo es definir una metodologia y unas herramientas útiles para la construcción de interfaces en lenguaje natural para realizar con diferentes aplicaciones diferentes interfaces en castellano con los que efectuar tareas como asimilar conocimiento a partir de textos del campo de aplicacion, recuperar información conocida por un sistema o bien explicación del razonamiento de generación de respuestas.

La estructura de un sistema basado en el conocimiento con el sistema Sirena es:



En este trabajo se presentan los aspectos linguisticos de este tipo de sistemas por lo que tareas como coherencia en la adquisición de conocimiento, tipos de inferencias realizadas o de representación seleccionada no se describiran ni justificaran.

El formalismo gramatical utilizado se basa en las Gramáticas de Metamorfosis definidas por Colmerauer (1975) y a los últimos trabajos de R. Pasero en cuanto a la estrategia que dirige el análisis.

La definición de la gramática, es decir de los tipos de frases a reconocer, pretende ser simple pero al mismo tiempo suficientemente amplia para abarcar frases consideradas estandar en la comunicación con un ordenador o bien básicas para representar el conocimiento disponible sobre un determinado dominio de aplicación.
Se dispone de un conjunto base de palabras consideradas como pertenecientes a cualquier dominio de discurso, como por ejemplo las que introducen a conectores (y, o, entinces ...) o algunos verbos como ser, estar y otros. A partir de este "diccionario básico" será posible definir suconjunto del castellano adecuado a la aplicación en forma interactiva.

Dado que no existe una elección de significado de las frases adecuada a toda aplicación se ha optado por la representación lógica mas general en una lógica basada en el cálculo de predicados y a continuación efectuar una "traducción" a la representación lógica de la aplicación.

Una vez definida la gramática y la representación semántica de cada uno de los componentes gramaticales la implementación es practicamente inmediato como se verá a continuación en el apartado uno de este capítulo al realizar completamente la implementación de una gramática prototipo.
La ampliación de la gramática con nuevas estructuras de frases (que pueden utilizar partes ya definidas) se realiza de forma sencilla ya que en general, cada tipo de frase se define con una sola regla Prolog que ha de ser añadida al programa. Ademas cualquier restricción que se desee expresar suele implicar la introducción de un nuevo argumento o bien (si es necesario más que la unificación) la introducción en el módulo de análisis, una secuencia de condiciones a verificar.
Se ha intentado en todo momento de la implementación que en las reglas de la gramática sólo aparecezcan caracteristicas gramaticales.

Convencionalmente el estudio de cadenas de simbolos con objeto de comprobar si son frases pertenecientes al lenguaje deseado consta de un análisis lexico-morfologico (forma de las palabras, significación y etimologia), de un análisis sintáctico (coordinación y union de las palabras del lenguaje para formar frases), de un análisis semántico (significado de la frase) y de un análisis praqmático (utilización de los resultados de los análisis anteriores). En el sistema construido aparecen estas fases en la forma en que se describe brevemente en el apartado uno.

Una vez que se ha obtenido una respuesta a la pregunta solicitada existe un módulo de explicación a partir de la traza o razonamiento seguido por el sistema basado en el conocimiento. No utiliza este módulo la misma metodologia que los módulos de adquisición de conocimiento o preguntas al sistema, pero se deja para un trabajo futuro las posibilidades que se vislumbran en este momento.

La programación de Sirena se ha realizado en el lenguaje de programación lógica Prolog, elegido para el desarrollo por ser considerado el más adecuado tanto a los formalismos gramaticales utilizados como al tipo de posibles sistemas informáticos que utilicen los interfaces construidos a partir de este sistema general.
El lenguaje de programación lógica Prolog (herramienta básica) adolece de ciertas ventajas deseables e incluso imprescindibles como procedimientos de gestion de recursos del lenguaje o del ordenador que no permiten una gran eficiencia en la ejecución. Sin embargo las ventajas se encuentran en el momento de diseño y depuración.

La implementación ha sido realizada sobre un IBM PC de 512K utilizando el lenguaje de programación Prolog II (v2) del GIA de la Facultad de Ciencias de Marsella (Francia) [Giannesini et all, 85], [Colmerauer, 83]. Es transportable a un ordenador VAX y por lo tanto el sistema total.

Debido a las características del lenguaje de programación ha sido posible realizar diferentes tipos de implementaciones para distintos tipos de requerimientos. Es decir el manejo complejo de los argumentos de los predicados de las clausulas,la introducción de predicados en las clausulas que definen a la gramática o de predicados en las clausulas que definen la estrategia de análisis y la facilidad de "congelar" resoluciones ha permitido expresar, representar y computar diferentes tipos de restricciones del proceso. Ademas se ha conseguido aislar cada una de las restricciones de forma que las restricciones gramaticales aparecen en las reglas de la gramática y las de la estrategia de construcción en las reglas del algoritmo de análisis.

Existen problemas con largas sesiones debidas a la implementación del lenguaje y es corriente la aparición de mensajes de error del tipo "SUBSTITUCION ... OVERFLOW", teniendo que volver a comenzar la tarea.

Se pasa a continuación a describir el sistema.

V.1 METODO DE ANALISIS DEL LENGUAJE

En este apartado se va a describir la parte de análisis del castellano del sistema Sirena.

Analisis léxico-morfológico: No existe análisis léxico propiamente dicho, pues dada una frase se considera una palabra a la cadena de simbolos entre dos blancos y se aceptaran todas aquellas que esten definidas en el diccionario del sistema.
El sistema dispone de un diccionario llamado "diccionario básico" que constarà de un conjunto elemental de palabras como son articulos, conjunciones, verbos, y otras palabras consideradas muy utilizadas en general en la construcción de frases en castellano.

El diccionario que se utilizará en un determinado sistema será creado añadiendo al diccionario general el conjunto de palabras propio al dominio de la aplicación. Esta tarea será realizada interactivamente por cualquier usuario y en cualquier momento siguiendo los pasos indicados por el sistema. Es decir, se parte de un conjunto minimo de palabras, suficiente para trabajar con textos del dominio y utilizar el mecanismo de ampliacion del diccionario (diccionario dinámico) para asimilar nuevas palabras o significados de palabras ya conocidas. De esta forma no hay necesidad de almacenar previamente al uso del interfaz grandes diccionarios.

Antes de comenzar el proceso de análisis de una frase se comprueba que todas las palabras que la forman pertenezcan al diccionario. En caso negativo se pide al usuario una solución entre varias posibles.

Analisis sintáctico: La sintaxis se puede considerar el motor básico en el sistema ya que dirige los procesos de reconocimiento y generacion de frases y su nivel de

caracterizacion es utilizado para la definicion de las palabras en el diccionario del sistema.
El reconocimiento de una frase de entrada como perteneciente al lenguaje definido se basa en la comparación de izquierda a derecha de las palabras de la frase con la frontera (conjunto de nodos terminales) de los diferentes arboles sintàcticos construidos automaticamente a partir de la gramática. Si la frase coincide con la frontera de alguno de ellos entonces el análisis habra terminado con exito, y este serà el arbol que representa su estructura sintáctica. En otro caso (no se corresponde con una estructura sintáctica correcta) la frase no será reconocida como perteneciente al lenguaje. Este metodo de análisis es no determinista con lo que será posible obtener todos los análisis posibles de la frase de entrada.

La definicion de las estructuras sintácticas se hace directamente con clausulas de Horn. En general es posible afirmar que cada estructura sintáctica se representa con una clausula de Horn. Este formalismo hace que la definicion de nuevas estructuras sintácticas sea facilmente realizable, ya que no es necesario estudiar las definiciones de estructuras ya existentes, aunque puedan ser utilizadas para consequir mayor eficacia y sintetizacion de la gramática.

En esta fase se dispone de un mecanismo de recureración de errores de concordancia de genero y número entre las palabras que componen la frase. En caso de ser detectado un error de este tipo se presentan al usuario todas las posibilidades de escribir la frase correctamente y una vez seleccionada una por él se continuará el análisis (se detalla en el apartado 2 de este capítulo).

Analisis semántico: La semántica se define basandose en la tecnica utilizada por A. Colmerauer. El significado de una frase podrá ser expresado en cálculo de predicados, cálculo concepto, atributo valor o cálculo proposicional.
El sistema permitirá la utilización de cualquiera de los tres

tipos de semántica. En este punto es el tipo de aplicación la que decide que formalismo es el más adecuado (el apartado tres de este capítulo trata la semántica o significado de las estructuras).

La formula que representará la semántica de la frase se va contruyendo paralelamente a la estructura sintáctica. La no posibilidad de asignar una estructura semántica correcta a una frase da lugar tambien a no considerar la frase como correcta. Es por esto que el tipo de análisis realizado es sintáctico semántico.

Como trabajo futuro se va a analizar la posibilidad de dotar a cada estructura sintáctica de una semántica que venga representada por un conjunto de significados diferentes y que sea el contexto en el que se efectua el análisis el que decida cual de ellos es el válido en ese momento. Tambien se pretende estudiar la conveniencia de dotar al sistema de análisis de un mecanismo de aprendizaje que permita tener en cuenta las posibles correcciones semánticas efectuadas por usuarios incorporando esta nueva informacion (dependiente del contexto) sobre el análisis.

Analisis pragmatico: El análisis pragmatico tiene por objeto decidir el tipo de utilización que se ha de dar al contenido de la frase o al resultado del análisis total de la frase.
Algunos trabajos actuales se basan en la teoria de Searle. En ella el acto de la palabra es la unidad mínima de comunicación y pretende deducir la intención del hablante al hablar. Ha categorizado ciertos actos verbales en los que aparecen el hablante y el agente estableciendo condiciones de interpretación (orden, ruego, ...). La formalización de esta teoria se hace a partir de tecnicas de planificación, expresando los actos verbales como planes y los procesos de comunicación como estrategias de razonamiento para construir o comprender planes.

El sistema no presenta ningun tipo de estudio pragmático, una vez analizada la frase de entrada su semántica asociada es ejecutada por un sistema de deducción y realiza lo que "la frase" indica sobre la base de conocimiento disponible.

Como ya se ha indicado, se va a desarrollar detenidamente una gramática prototipo pero incluida en la final. La única variación de implementación de la gramática (es decir del análisis sintáctico y el semántico) consiste en la definición de las palabras en el diccionario.
Por razones de eficiencia se ha optado por no tener almacenadas todas las palabras a partir de su categoria sintáctica (como en el prototipo) sino a partir de la palabra misma.

A continuación se realiza la implementación de una gramática prototipo y de las fases de análisis sintáctica y semántica.

Sea la gramática siguiente escrita en BNF y con semántica definida entre corchetes de cuyo significado no es descrito en este momento (se describe detalladamente en el apartado tres).

"Estructuras gramaticales"

{{<i,o1>,o},<i,o1>, o}
<frase> ::= <grupo nominal> <grupo verbal>
{i, {<i,o1>,o1}}
<grupo nominal> ::= <nom propio>
{i, {<i,o1>,o1}}
<grupo nominal> ::= <preposicion> <nom propio>
{{<j,o2>,<i,o1>,o},<j,o2>, {<i,o1>,o}}
<grupo nominal> ::= <grupo articulo> <nom comun>
{{<i,<j,o2>>,<i,o1>,o3},<i,<j,o2>>,{<j,o3>,o}, {<i,o1>,o}}
<grupo nominal> ::= <grupo articulo> <nom comun>
<grupo nominal>

{<i,o1>, <i,o1>}

<grupo verbal> ::= <verbo ser> <atributo>

{<i,<j,o2>>,{<j,o2>,o1}, <i,o1>}

<grupo verbal> ::= <verbo t1> <grupo nominal>

{{<i,o1>,o1}, <i,o1>}

<atributo> ::= <adjetivo>

{<i,o1>, <i,o1>}

<atributo> ::= <articulo> <nom comun>

{<i,<j,o2>>,{<j,o2>,o1}, <i,o1>}

<atributo> ::= <articulo> <nom comun> <grupo nominal>

{o, o}

<grupo articulo> ::= <preposicion> <articulo> / <articulo> /
<preposicion>

{<i,o1>,<i,o2>,existe(i,conj(o1,o2))}

<grupo articulo> ::= <art contracto>

{<i,o1>,<i,o2>,existe(i,conj(o1,o2))}

<articulo> ::= <art definido (sin)> / <art indefinido>

{<i,o1>,<i,o2>,todo(i,dis(no(o1),o2))}

<articulo> ::= <art definido (plu)>

"Diccionario"

<nom propio> ::= Manuel / Alicia / Juan / Pilar / Prolog

<nom comun> ::= niño / niña / niños / niñas /
hermano / hermana / hermanos / hermanas/
hombre / mujer / animal / animales

<adjetivo> ::= alto / alta / altos / altas /
bonito / bonita / bonitos / bonitas

<verbo ser> ::= es / son

<verbo t1> ::= estudia / estudian

<preposicion> ::= de / a

<art contracto> ::= al / del

<art definido> ::= el / la / los / las

<art indefinido> ::= un / una / unos / unas

Frases del lenguaje de esta gramática son:

Manuel es alto.

con semántica:alto(Manuel)

Manuel es un niño.

con semántica: niño(Manuel)

Manuel estudia los animales de Alicia.

con semántica:

existe(x,conj(animal(x,Alicia),estudia(Manuel,x)))

las hermanas de Juan estudian Proloq.

con semántica:

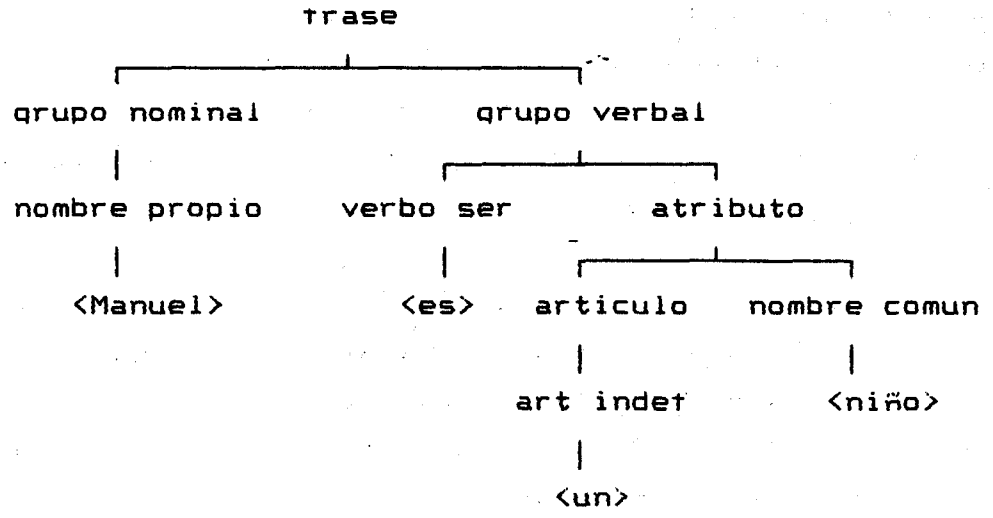
todo(x,dis(no(hermana(x,Juan)),estudia(x,Proloq)))

Siendo la semántica de los nombre propios el objeto del dominio asociado a ese nombre, la de los nombres comunes un nombre de predicado y la de los verbos tipo uno un predicado binario (para indicar el objeto sujeto y el objeto directo).

En la definición anterior no han aparecido las restricciones de género y número del castellano salvo en el articulo definido por afectar a la semántica. Se tendran en cuenta posteriormente en otra fase del desarrollo.
Por ello en este momento una frase de esta gramática es "Manuel es una niña" o bien "las niñas es bonita".

Una vez conocida la gramática que se va a tratar en este apartado se comienza con la realización de la implementación sintáctica. Falta indicar cual será la representación Proloq de las frases y cual su estructura sintáctica.

Se representa las frases con la lista Proloq de palabras que la forman:
Manuel.es.un.niño.nil
Su sintaxis es el arbol construido a partir de las reglas que la analizan:



Las palabras en castellano se encierran entre corchetes "<" y ">" para indicar que son nodos terminales.

Una regla de la gramática se entiende como un proceso de reescritura o sustitución con unificación de una parte de la regla por la otra. Por ejemplo, la primera regla que describe las frases del lenguaje descrito por la gramática, indica que se tiene una frase del lenguaje si es posible encontrar en ella un grupo nominal seguido de un grupo verbal o bien, si se encuentran estos dos componentes en la frase de entrada entonces se tiene una frase del lenguaje.

Cada regla en BNF se representa con una regla Prolog de la manera siguiente:

- A cada símbolo le corresponde un nombre de predicado con un argumento que indicará la estructura sintáctica asociada (que será construida a partir de los constituyentes lo definen en la regla BNF). Por ejemplo frase(fr(x,y)) siendo frase el nombre de predicado asociado a frase, parte izquierda de la primera regla en BNF, y fr(x,y) la estructura sintáctica asociada (que en prolog es la estructura de arbol y se entiende como un argumento).

- A cada regla en BNF se asocia una regla Prolog con los constituyentes asociados segun el punto anterior. Por ejemplo

la primera regla será:

```
frase(fr(x,y)) -> grupo-nom(x) grupo-ver(y);
```

que indica que una algo será frase si es posible asociarle la estructura fr(x,y) donde x es la estrucutra sintáctica de un grupo nominal e y de un grupo verbal.

- A cada palabra del diccionario se le asocia una regla Prolog con parte derecha vacia y parte izquierda con nombre de predicado la categoria sintáctica y argumentos la propia palabra, información sobre su genero y número y finalmente semántica asociada. Por ejemplo:

```
nc(niño,mas.sin,niño) ->;
np(Manuel,mas.sin,Manuel) ->;
ver-t1(estudia,mas.sin,<i,<j,estudia(i,j)>>) ->;
prep(de,x,y) ->; (no tiene genero, numero ni semántica)
```

La gramática en clausulas Prolog será según los puntos anteriores:

```
frase(fr(x,y)) -> grupo-nom(x) grupo-ver(y);

grupo-nom(gn(x)) -> preposicion(x);
grupo-nom(gn(x,y)) -> preposicion(x) nom-propio(y);
grupo-nom(gn(x,y)) -> grupo-art(x) nom-comun(y);
grupo-nom(gn(x,y,z)) -> grupo-art(x) nom-comun(y)
                        grupo-nom(z);

grupo-ver(gv(x,y)) -> verbo-ser(x) atributo(y);
grupo-ver(gv(x,y)) -> verbo-t1(x) grupo-nom(y);

atributo(att(x)) -> adjetivo(x);
atributo(att(x,y)) -> articulo(x) nom-comun(y);
atributo(att(x,y,z)) -> articulo(x) nom-comun(y)
                        grupo-nom(z);

grupo-art(gr-art(x,y)) -> preposicion(x) articulo(y);
grupo-art(gr-art(x)) -> preposicion(x);
```

```
grupo-art(gr-art(x)) -> art-contracto(x);
grupo-art(gr-art(x)) -> articulo(y);

articulo(art(x)) -> art-definido(x);
articulo(art(x)) -> art-indefinido(x);

nom-propio(np(<x>)) -> np(x,y,z);
nom-comun(nc(<x>)) -> nc(x,y,z);
adjetivo(adj(<x>)) -> adj(x,y,z);

verbo-ser(ser(<x>)) -> ver-ser(x,y,z);
verbo-t1(ver(<x>)) -> ver-t1(x,y,z);

preposicion(prep(<x>)) -> prep(x);
art-contracto(art-c(<x>)) -> art-c(x,y,z);
art-definido(art-def(<x>)) -> art-def(x,y,z);
art-indefinido(art-indef(<x>)) -> art-indef(x,y,z);

"Diccionario"

np(Manuel,mas.sin,Manuel) ->;
np(Alicia,fem.sin,Alicia) ->;
np(Juan,mas.sin,Juan) ->;
np(Pilar,fem.sin,Pilar) ->;
np(Prolog,mas.sin,Prolog) ->;
nc(niño,mas.sin,niño) ->;
nc(niños,mas.plu,niño) ->;
nc(niña,fem.sin,niña) ->;
nc(niñas,fem.plu,niña) ->;
nc(hermano,mas.sin,hermano) ->;
nc(hermanos,mas.plu,hermano) ->;
nc(hermana,fem.sin,hermana) ->;
nc(hermanas,fem.plu,hermana) ->;
nc(hombre,mas.sin,hombre) ->;
nc(mujer,fem.sin,mujer) ->;
nc(animal,mas.sin,animal) ->;
nc(animales,mas.plu,animal) ->;
```

```
adj(bonito,mas.sin,bonito) ->;
adj(bonitos,mas.plu,bonito) ->;
adj(bonita,fem.sin,bonita) ->;
adj(bonitas,fem.plu,bonita) ->;
adj(alto,mas.sin,alto) ->;
adj(altos,mas.plu,alto) ->;
adj(alta,fem.sin,alta) ->;
adj(altas,fem.plu,alta) ->;
ver-ser(es,x.sin,y) ->;
ver-ser(son,x.plu,y) ->;
ver-t1(estudia,x.sin,estudia) ->;
ver-t1(estudian,x.plu,estudia) ->;
prep(de) ->;
prep(aa) ->;
art-c(al,mas.sin,x) ->;
art-c(del,mas.sin,x) ->;
art-def(el,mas.sin,x) ->;
art-def(la,fem.sin,x) ->;
art-def(los,mas.plu,x) ->;
art-def(las,fem.plu,x) ->;
art-indef(un,mas.sin,x) ->;
art-indef(una,fem.sin,x) ->;
art-indef(unos,mas.plu,x) ->;
art-indef(unas,fem.plu,x) ->;
```

Con objeto de no perder generalidad en algunas reglas del diccionario se ha indicado con una variable la no existencia de ese elemento. Por ejemplo los verbos no tienen género luego sólo aparece en su lugar una variable.

Nota sobre el lenguaje de programación:
Los caracteres dobles no son admitidos (ñ sustituida por n, no hay acentos).
Las letras mayusculas y minusculas son caraceters diferentes por lo que las palabras han de ser escritas en todas las sesiones de la misma forma que en el programa.
Las palabras del castellano formadas por una sola letra han

de ser duplicadas para que no se confundan con una variable (a preposición por aa).

Como ya se ha dicho una frase es la lista Prolog de las palabras que la componen. Además ha de ser la lista de izquierda a derecha de las hojas terminales de la estructura sintáctica o arbol sintáctico asociado a la frase (frontera del arbol).
Luego una estrategia de análisis es construir un arbol sintáctico de arriba a abajo (estratégia predefinida Prolog) y comprobar si su frontera (lista de izquierda a derecha de palabras que etiquetan nodos terminales) es exactamente la lista de entrada. Expresado en Prolog:

```
analizar(l,a) -> frase(a) frontera(a,l);
```

es decir si "a" es un arbol sintáctico con frontera la lista "l" entonces el resultado de analizar la secuencia de palabras "l" es correcto, luego "l" es una frase perteneciente al lenguaje descrito por la gramática con estructura sintáctica "a".

En este momento se dispone de un programa Prolog que representa las reglas de la gramatica y de un módulo que dirige el proceso de análisis (regla analizar junto con la definición Prolog de "frontera").
El problema de esta estrategia, la mas habitual en este tipo de sistemas, es que es ineficaz por calcular todos los arboles sintácticos sin tener en cuenta información aportada por la frase de entrada y ademas, si la gramática es infinita el análisis puede no pararse independientemente de la frase de entrada.
Para evitar ambos problemas se modifica la estrategia anterior de forma que "cada vez que se llega a una hoja) se comprueba si es la correspondiente en la lista de palabras de entrada".
La descripción del módulo "frontera" en las dos versiones se

encuentra en el apartado siguiente de este capítulo.
La nueva implementación obliga a que el objetivo Prolog "analizar" invierta su lista de objetivos de la parte derecha:

```
analizar(l,a) -> frontera(a,l) frase(a);
```

Algunos ejemplos de análisis con este programa:

```
>anal;
DIGAME UNA FRASE
las hermanas de Juan estudian Prolog.
SINTAXIS:
fr(gn(gr-art(art(art-def(<las>))),nc(<hermanas>),
      gn(prep(<de>),np(<Juan>))),
   gv(ver(<estudian>),gn(np(<Prolog>))))
()
>anal;
DIGAME UNA FRASE
Manuel es alto.
SINTAXIS:
fr(gn(np(<Manuel>)),gv(ser(<es>),att(adj(<alto>))))
()
>anal;
DIGAME UNA FRASE
las nino son bonitas.
SINTAXIS:
fr(gn(gr-art(art(art-def(<las>))),nc(<nino>)),gv(ser(<son>),a
tt(adj(<bonitas>)) ))
()
>
```

Para que no sea posible el último análisis (en castellano no es correcto) se modifica el programa anterior para tener en cuenta las restricciones de género y número. Al ser una caracteristica gramatical se representa en las reglas introduciendo un nuevo argumento: una lista "g.n" con "g"

indicando el género y "n" el número. Por ejemplo:

```
grupo-nom(g.n,gn(x,y,z)) -> grupo-art(g.n,x) nom-comun(g.n,y)
                            grupo-nom(g'.n',z);
```

Luego el género y número del grupo de articulos y del nombre comun han de ser iguales no siendo así el del grupo nominal (constituyentes de la regla).
Debido a que la unificación es una operación predefinida del lenguaje de implementación se escribe esta regla:

```
grupo-nom(t,gn(x,y,z)) -> grupo-art(t,x) nom-comun(t,y)
                          grupo-nom(t',z);
```

La regla que da paso al programa queda:

```
analizar(l,a) -> frontera(a,l) frase(t,a);
```

A continuación se modifican las reglas Prolog de la gramática para construir el significado de las frases. De nuevo es una caracteristica que ha de aparecer en las reglas por ser construida la estructura semántica paralelamente a la estructura sintáctica. Un fracaso de construcción de significado produce tambien un fracaso en el análisis sintáctico (como cabria esperar por la implementación paralela efectuada). Es por esto por lo que decimos que el sistema realiza un análisis sintáctico-semántico.

Se introduce un nuevo argumento en los predicados de las reglas Prolog (o varios según el tipo de estructura semántica). Como ya sería al menos el tercer argumento y con objeto de claridad se van a agrupar el argumento sintáctico y el de concordancia con la estructura Prolog II de corchetes "<" ">". Por ejemplo la tercera regla de grupo nominal será:

```
grupo-nom(<t,gn(x,y,z)>) -> grupo-art(<t,x>)
                            nom-comun(<t,y>)
```

```
                                grupo-nom(<t',z>);
```

Añadiendo los argumentos necesarios en cada uno de los predicados Prolog de las reglas de la gramática el programa reescribiendo la información que aparecia antes de cada regla en BNF se tiene:

```
frase(<t,fr(x,y)>,o) -> grupo-nom(<t,x>,<i,o1>,o)
                        grupo-ver(<t,y>,<i,o1>);

grupo-nom(<t,gn(x)>,<i,o1>,o1) -> nom-propio(<t,x>,i);
grupo-nom(<t,gn(x,y)>,<i,o1>,o1) -> preposicion(x)
                                    nom-propio(<t,y>,i);
grupo-nom(<t,gn(x,y)>,<i,o1>,o) ->
                        grupo-art(<t,x>,<j,o2>,<i,o1>,o)
                        nom-comun(<t,y>,<j,o2>);
grupo-nom(<t,gn(x,y,z)>,<i,o1>,o) ->
                        grupo-art(<t,x>,<j,o2>,<i,o1>,o3)
                        nom-comun'(<t,y>,<j,<k,o2>>)
                        grupo-nom(<t',z>,<k,o3>,o);

grupo-ver(<t,gv(x,y)>,<i,o1>) -> verbo-ser(<t,x>)
                                 atributo(<t,y>,<i,o1>);
grupo-ver(<t,gv(x,y)>,<i,o1>) -> verbo-t1(<t,x>,<i,<j,o2>>)
                                 grupo-nom(<t',y>,<j,o2>,o1);

atributo(<t,att(x)>,<i,o1>) -> adjetivo(<t,x>,<i,o1>,o1);
atributo(<t,att(x,y)>,<i,o1>) -> articulo(<t,x>,a,b,c)
                                 nom-comun(<t,y>,<i,o1>);
atributo(<t,att(x,y,z)>,<i,o1>) -> articulo(<t,x>,a,b,c)
                                nom-comun'(<t,y>,<i,<j,o2>>)
                                grupo-nom(<t',z>,<j,o2>,o1);

grupo-art(<t,gr-art(x,y)>,a,b,c) -> preposicion(x)
                                    articulo(<t,y>,a,b,c);
grupo-art(<t,gr-art(x)>,<i,o1>,<i,o2>,conj(o1,o2)) ->
                                    preposicion(x);
```

```
grupo-art(<t,gr-art(x)>,<i,o1>,<i,o2>,existe(i,conj(o1,o2)))
                                    -> art-contracto(<t,x>);
grupo-art(<t,gr-art(x)>,a,b,c) -> articulo(<t,x>,a,b,c);

articulo(<t,art(x)>,a,b,c) -> art-definido(<t,x>,a,b,c);
articulo(<t,art(x)>,a,b,c) -> art-indefinido(<t,x>,a,b,c);

nom-propio(<t,np(<x>)>,z) -> np(x,t,z);
nom-comun(<t,nc(<x>)>,<i,<z,i>>) -> nc(x,t,z);
nom-comun'(<t,nc(<x>)>,<i,<j,<z,i,j>>>) -> nc(x,t,z);
adjetivo(<t,adj(<x>)>,<i,<z,i>>,<z,i>) -> adj(x,t,z);

verbo-ser(<t,ser(<x>)>) -> ver-ser(x,t,z);
verbo-t1(<t,ver(<x>)>,<i,<j,<z,i,j>>>) -> ver-t1(x,t,z);

preposicion(prep(<x>)) -> prep(x);
art-contracto(<t,art-c(<x>)>) -> art-c(x,t,z);
art-definido(<t,art-def(<x>)>,<i,o1>,<i,o2>,
                                existe(i,conj(o1,o2))) ->
art-def(x,q.sin,z);
art-definido(<t,art-def(<x>)>,<i,o1>,<i,o2>,
                                todo(i,dis(no(o1),o2))) ->
art-def(x,q.plu,z);
art-indefinido(<t,art-indef(<x>)>,<i,o1>,<i,o2>,
                                existe(i,conj(o1,o2))) ->
art-indef(x,t,z);
```

Así la regla "analizar" es:

```
analizar(l,a,e) -> frontera(a,l) frase(<t,a>,e);
```

que expresa que una lista de palabras "l" es una frase del lenguaje si tiene por estructura sintáctica asociada el arbol "a" y semántica a "e".

Algunos ejemplos de análisis:

```
...
DIGAME UNA FRASE
las hermanas de Manuel estudian Prolog.
SINTAXIS:
fr(gn(gr-art(art(art-def(<las>))),nc(<hermanas>),gn(prep(<de>
),np(<Manuel>))), gv(ver(<estudian>),gn(np(<Prolog>))))
SEMANTICA
todo(x,dis(no(<hermana,x,Manuel>),<estudia,x,Prolog>))
...
DIGAME UNA FRASE
Manuel es alto.
SINTAXIS:
fr(gn(np(<Manuel>)),gv(ser(<es>),att(adj(<alto>))))
SEMANTICA
<alto,Manuel>
{}
>
```

V.2 ALGORITMO DE ANALISIS MODIFICADO

La modificación presentada al algoritmo utilizado en análisis del lenguaje natural [Giannesini, 85] permite expresar condiciones en el proceso general.

Se comienza presentando como dirige el proceso de análisis el algoritmo sin la modificación.
Acepta un arbol de derivación como estructura sintáctica de la frase si su frontera (lista de etiquetas de nodos terminales u hojas del arbol) coincide con la frase a analizar. De esta forma se van construyendo todos los arboles posibles pero en cuanto no coincide una hoja terminal con la correspondiente palabra en la lista frase de entrada se descarta este arbol y se construye el siguiente. La llamada a este proceso es:

frontera(a,frase-entrada) arbol-sintáctico(a)

El algoritmo que define la frontera del arbol "a" se basa en el predicado "hojas(x,l,l')" predicado que se "congela" o no ejecuta hasta que "x" tome valor (sea unificado con una estructura sintáctica en este caso). Este predicado se construye a su vez a partir de "hojas'(x,l,l')" que es verdad si "l'" es la lista de palabras obtenida a partir de las fronteras de los subarboles de "x".
Este algoritmo dirige el proceso de análisis de la frase a partir de la estructura sintáctica de la frase (al dirigir su construcción según la gramática implementada), con estrategia top-down, en profundidad y marcha atras.

El uso de este algoritmo exige la definición completa de la estructura sintáctica de los elementos gramáticales. Se recuerda que al ser la estructura semántica construida al mismo tiempo que la sintáctica y utilizar tambien la unificación (como herramienta del lenguaje) produce condiciones o restricciones a cumplirse en determinadas

ocasiones, restringiendo de esta forma la construcción de las estructuras de las reglas de la gramática. Es decir es un proceso basicamente sintáctico controlado por un modelo semántico.

Debido a que se tiene en cada momento control sobre los componentes de la estructura sintáctica de la frase se ha realizado la siguiente modificación.
La modificación consiste en construir no sólo la frontera del arbol a partir de sus componentes (nodos interiores) sino además otra estructura (de tipo lista) que expresa cierta información aparecida en los nodos del arbol y que a su vez va a ser tratada o procesada en el momento de construcción de la frontera o como en nuestro caso, al final del proceso total. De nuevo la implementación aprovecha el tratamiento de los argumentos de los predicados así como el predicado de "congelar" la resolución hasta un momento determinado por el propio programa.
En este trabajo se ha realizado así el tratamiento de la concordancia que han de verificar las palabras que forman una frase. No es una propiedad que pueda ser definida en forma general y standar (como frontera se define para arboles sin sub-arboles, con un sub-arbol, con dos, ...) sino que necesita una definición para cada tipo de componente sintáctico de la estructura (subarbol). Se describe a continuación esta realización.

En castellano una frase es sintácticamente correcta (según la definición de una gramatica) si es posible construir su arbol asociado y ademas si se verifica debidamente la concordancia de genero y numero entre las palabras que forman la frase.
Las restricciones que determina la concordancia entre los elementos de qun constituyente de la frase se expresan en la regla gramatical que lo define y han de ser verificadas para aceptarlo como valido.

La modificación que se propone en este método de análisis es "congelar" la verificación de la concordancia de los elementos del constituyente hasta que se haya realizado el análisis total de la frase. De esta forma una frase es sintácticamente correcta si es posible encontrar su arbol sintáctico tenga errores de concordancia o no. Caso de tenerlos se pasa a un proceso de recuperación que consiste en construir las frases correspondientes a la dada pero que verifican la concordancia entre las palabras. Por ejem:

La coyunturas es estable.
(error de concordancia)

Posibles soluciones:
La coyuntura es estable.
Las coyunturas son estables.

Siendo el usuario el que decide cual es la frase adecuada (que él deseaba analizar) ya que se ha considerado que el sistema no tiene suficiente información ni manera de decidir cual es la solución correcta.

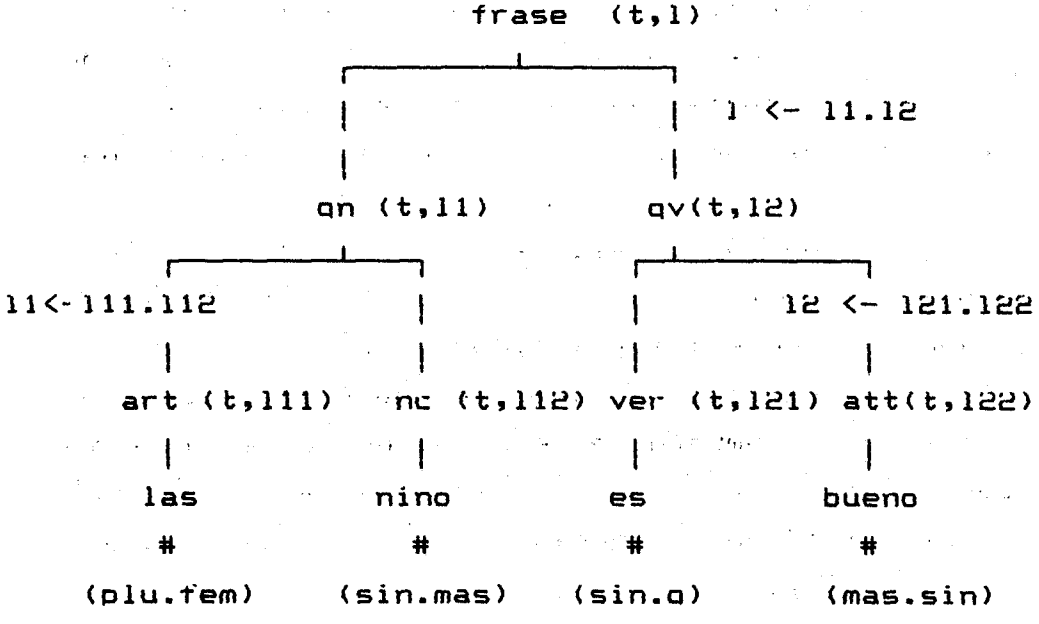
El motivo fundamental de introducir este tipo de recuperacion no solo es hacer mas "amable" el uso del interface al usuario sino tambien por razones semánticas. El tipo de análisis que se presenta en este trabajo se basa tanto en la sintaxis como en la semántica de las frase y por ello no es lo mismo que una frase tenga asociada una formula cuantificada universalmente (introducción por el articulo "las", por ejemplo) que existencialmente (introducido por el articulo "la"). No se desaba simplemente desechar la frase como no correcta y sin embargo en el caso de que el interfaz sea para comunicación con un sistema basado en el conocimiento el tipo de respuesta en los dos casos es completamente diferente.

La implementación de este nuevo proceso de detección de

error de concordancia ha obligado a :

1. En cada regla de la gramatica introducir una lista que indica para el constituyente que define cuales de sus elementos han de tener que restricciones en su concordancia.
2. Un modulo de detección-recuperación a partir de la lista de detección de errores de concordancia construida a lo largo del análisis de la frase.

Es el menor numero de aspectos necesarios de modificación para la realización de la detección y recuperación: una definición y un modulo que la maneje. La implementación se ha realizado asociando a cada constituyente dos indicadores de concordancia, la variable que contendra el genero y numero ("t") y la lista "l" que contendrá los errores (si los hay) de concordancia aparecidos en alguno de sus elementos. Por ejemplo en el caso de la definición de frase se exige que tenga un genero y numero "t" y que sus constituyentes grupo nominal y grupo verbal tengan el mismo "t". En el ejemplo siguiente aparecen errores tanto de genero como de número.



Se describe a continuación la implementación realizada

para la detección y recuperación de este tipo de error:

1. Se introduce en las reglas gramáticales las restricciónes que han de verificar los constituyentes así como una estructura de lista que contendrá los errores aparecidos. En la parte de restricciones sintácticas de cada predicado aparece una lista de variables indicando que en la primera variable apareceran los errores de los constituyentes y del propio predicado. Por ejemplo:

```
gr-nom(<t,1.11.12.<13>>,x,y,z) si
        gr-art(<t,11>,x)
    y   nombre-comun(<t,12>,y)
    y   gr-nom(<t',13>,z);
```

El proceso de construcción de "l" es dirigido por el algoritmo de análisis modificado y realizado por el módulo de detección-recuperación de errores de concordancia.

2. En cada una de las clausulas que definen el algoritmo de análisis se ha de llamar al módulo de construcción de la lista de errores (detección-recuperación de errores de concordancia):

```
hojas(<c,l-r,x1,x2>,l,l1) si
        hojas(x2,l,l2)
    y   hojas(x1,l2,l1)
    y   detec(x,l-r);
```

Siendo "c" el nombre del grupo a construir (información que ya aparecía), "l-r" la lista necesaria para detectar y reconstruir los errores aparecidos en el grupo en estudio, y "x1", "x2" los constituyentes.
El predicado "hojas(a,b,c)" es cierto cuando las hojas del arbol "a" son las comprendidas entre "b" y "c".
El predicado "detec(c,l-r)" retarda el estudio de la concordancia hasta finalizar el análisis y construye la lista "l", posiblemente vacia.

De esta forma al terminar el análisis se tendrá un conjunto de restricciones para evaluar del tipo:

```
detec(c,<q.n,l.l1.l2.<l3>>)
'o'
unif(t,t',x,l)
```

indicando (en el primer tipo de restricción) que el genero "q" y el número "n" del constituyente "c" han de ser los de los elementos con lista de error "l1" y "l2". El último elemento del constiuyente "c" puede no tener el mismo genero y numero luego su lista de errores "l3" no depende de ellos. Cuando se llega a una estructura terminal, en este caso una palabra de la frase, la detección del error se reduce al intento de unificación del genero y numero real con el esperado. Si coinciden entonces la lista toma el valor de "sintaxis correcta" y si no es así toma el valor del predicado "error(s,palabra) donde s es (gen,num,num.gen). Esto se realiza con llamadas del segundo tipo de restricciones a evaluar (predicado unif).

Una vez acabado el análisis y esta fase si la lista de errores del nodo raiz de la estructura de la frase es distinta de "sintaxis correcta" se pasa a la fase de reconstrucción del error/es aparecido. La recuperación del error se realiza por reemplazo de las palabras erroneas con las correctas y se vuelve a comenzar con esta frase el análisis por posibles variaciones en sus estructuras.

A continuación se muestran algunos ejemplos de una sesión con el sistema.

```
...
DIGAME UNA FRASE
el limites es bajas y la coyuntura es favorable.
ERROR DE CONCORDANCIA EN EL genero Y numero
el limite es bajo y la coyuntura es favorable
```

ES LA FRASE CORRECTA ? no

los limites son bajos y la coyuntura es favorable

ES LA FRASE CORRECTA ? si

los limites son bajos y la coyuntura es favorable

...

DIGAME UNA FRASE

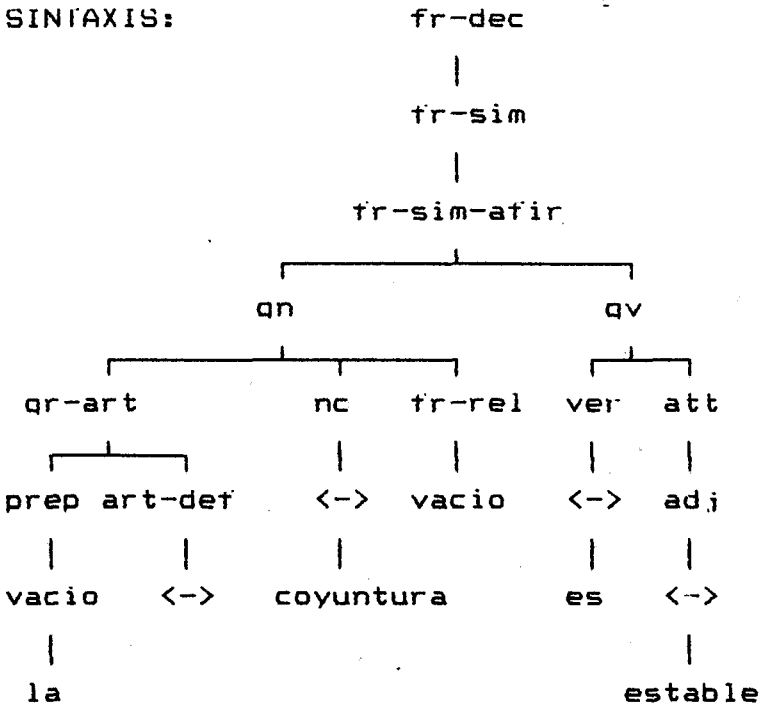
la coyunturas es estable.

ERROR DE CONCORDANCIA EN EL numero

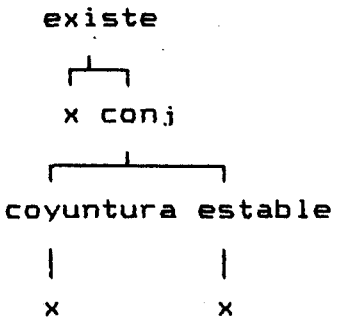
la coyuntura es estable

ES LA FRASE CORRECTA ? si

SINTAXIS:



SEMANTICA:



frases correctas fuese adecuada para su posterior proceso con el sistema basado en el conocimiento (tipo base de datos, sistema experto, ...) sin perder en ninqun momento la qeneralidad y posibilidad de corrección. La semántica utilizada esta basada en el cálculo de predicados y existe un módulo de transformación a lóqica proposicional, clausulas de Horn o lóqica de concepto atributo valor para los casos en los que el sistema con el que se comunica utiliza alquno de estos tipos de representación.

V.3.1 INTERPRETACION SEMANTICA

Utilizar la lógica como lenguaje semántico permite un gran poder expresivo y una buena claridad. La semántica puede ser directamente ejecutable una vez construida si lo permite el sistema basado en el conocimiento anexo, es decir si es posible el acceso a él mediante sentencias lógicas. Realizar un interprete intermedio entre la semántica construida en cálculo lógico y un lenguaje de acceso (por ejemplo a una base de datos) no es un trabajo dificil ya que es de la misma naturaleza y normalmente más restringido que cualquier tratamiento con el lenguaje natural.

La semántica utilizada en el sistema final se basa en los puntos o hipótesis que se presentan a continuación definidos a partir de las consideraciones semánticas dadas por Colmerauer [Colmerauer 77] y por R. pasero [Pasero 82] para el frances. Se describen las realizadas para el castellano. Las diferencias la mayor parte de las veces esta en la forma y no en el fondo. Es un estudio y delimitación de un subconjunto del lenguaje natural util como punto de base para consulta y creación de sistemas basados en el conocimiento.

La definición semántica de los elementos de la gramática incluye dos aspectos: La definición del tipo de estructura (formula) asociada a cada grupo y su forma de cálculo a partir de las estructuras de sus constituyentes. Para cada palabra se dispone de una tupla que la define y en ella existe un elemento que describe su semántica, mientras que la de los grupos de palabras aparece en la regla gramatical que le define.

Las hipótesis de representación de frases o de sus componentes son (se muestra un ejemplo de representación y a continuación la hipótesis realizada):

"Juan es un hombre"

hombre(Juan)

"Juan es hermano de Alicia"

hermano(juan, Alicia)

HIP1: Todo verbo, adjetivo o nombre comun crea un nombre de predicado (relación) con n argumentos

"Un niño viene"

existe x(niño(x) /\ viene(x))

HIP2: A cada articulo le corresponde un cuantificador que afecta a una formula formada por dos subformulas.

"ningun hombre tiene un rabo"

no(existe x (hombre(x) /\ (existe y (rabo(y) /\ pertenece(y,x)))))

HIP3: La cuantificación introducida por el articulo del sujeto de un verbo domina a la cuantificación introducida por el complemento del verbo (se excluyen frases adverbiales). El cambio de una frase de activa a pasiva produce la inversión de la jerarquia de los cuantificadores.

"las ventanas de las casas son grandes"

paratodo(x,(casa(x) -> (paratodo(y,(ventanas(y,x) -> grandes(y)))

HIP4: Cuando aparece un nombre comun y un complemento de ese nombre, la cuantificación introducida por el articulo del complemento domina la cuantificación introducida por el articulo del nombre.

"Juan tiene los libros de las asignaturas de la carrera"

existe(x,carrera(x) /\ paratodo(y,asignaturade(y,x) -> paratodo(z,librode(z,y) -> tiene(Juan,z))))

HIP5: Siempre que un verbo, un adjetivo o un nombre tenga dos complementos, la cuantificación se hace en el sentido inverso a su aparición, es decir, el complemento más a la derecha genera una cuantificación que domina a la cuantificación generada por el otro complemento.

"Un perro cojo se interesa por el encaje de bolillos"

paratodo x (perro(x) /\ cojo(x) --> ieb(x))

existe x (perro(x) /\ cojo(x) /\ ieb(x))

El tipo de ambiguedad de la frase anterior que sólo podría ser eliminada por el contexto en este trabajo se elimina asumiendo que en castellano el articulo defefinido singular y el articulo indefinido introducen una cuantificación existencial.

"Manuel no solicita un credito"

paratodo(x,animal(x) --> no(estudia(Manuel,x)))

HIP6: La negación que acompaña a un verbo es representada por la conectiva "no" aplicada a la formula introducida por su grupo verbal.

"el hombre que viene es joven"

existe(x,(hombre(x) /\ viene(x)) /\ joven(x))

HIP7: Las frases relativas se modelizaran como una frase usual pero el pronombre relativo es sustituido por la variable correspondiente y toda la modelización pasa a ser el segundo elemento de la conectiva "y" que aparece en lugar del nombre de la frase principal.
(nunca la cuantificación que pueda aparecer en la frase relativa pasa por delante del articulo de la frase principal)

Claramente no todos los tipos de estructuras de frases del castellano han sido definidas por los puntos anteriores.

Asi mismo tampoco se ha tratado con conjuntos que verifican propiedades sino con individuales pero si se considera que un individuo (de cualquier tipo) es un conjunto de un solo elemento puede ser añadida la siguiente hipotesis:

HIP8: Las propiedades n-arias introducidas por los verbos, nombres y adjetivos se entienden aplicadas a conjuntos de individuales.

Hay trabajos de tratamiento del lenguaje que sólo permiten el uso de palabras en plural si se utiliza un nuevo formalismo semántico que tiene en cuenta los conjuntos de individuales o de objetos. Para ello introducen el concepto de "cardinalidad" de un conjunto:

card(C,P) = n El cardinal del conjunto C que se verifica en P es n

Ejemplos:

"Dos perros ladran"	card(C,(perro(C) y ladra(C))) =2
"Unos hombres hablan"	
"Un hombre habla"	card(C,hombre(C) y habla(C))) >0 existe(x,(hombre(X) y habla(x)))
"La niña llora"	card(C,niña(C)) =1 and card(C,(niña(C) y llora(C))) =1 existe(x,(niña(x) y llora(x)))
"Los niños juegan"	card(C,niño(C)) >1 and card(C,(niño(C) y no(juega(C)))) =0 paratodo(x,(no(niño(x)) o juega(x)))

Una descripción completa del uso de conjuntos para expresar el plural con este formalismo puede ser encontrada en [Bernth, 85]

NOTA:

La notación anterior es modificada de la siguiente forma:

Objetos = (Variables y Constantes)

Estructura básica = <i1,<i2,...,<in,F>...>

i1, ...in son objetos sobre los que se construye la formula F

Notación equivalente a F(i1, ...,in)

Son formulas :

<Juan, hombre(Juan)>

<i,<j,hermano(i,j)>>

<i,<j,conj(hombre(i),dis(no(mujer(j)),casado(i,j)))>>

Antes de cada regla sintáctica de la gramática aparece el conjunto que define la semántica de la regla: (e1,e2,...en, e) que indica que a partir de las estructuras e1,...en (semántica de los componentes de la parte drcha de la regla) se construye e semántica del elemento parte izda.

Estructura semántica en lógica proposicional

El elemento básico de la lógica proposicional es la "proposición" definida como la unidad mínima de información sobre la cual puede decidirse la verdad o falsedad (es representada por letras minusculas). Una formula proposicional definida sobre el alfabeto { y, o, no, ->, {a,b,c, ...}} es una letra proposicional o bien, siendo A y B formulas proposicionales, (A y B), (A o B), (A -> B), (no A) y ninguna mas. El ambito de cada conectiva en cada formula viene determinado por la siguiente jerarquia de prioridades: { -> }, { y, o } y finalmente { no }. La conectiva {<->} se entiende como la conjunción de dos implicaciones simples pudiendo ser así consideradas formulas que la contienen como pertenecientes a la lógica proposicional.

Son proposiciones las frases simples (frases formadas por qn, y qv como por ejemplo "Manuel es un niño"). Las conectivas representan los tipos de conexiones entre frases proposicionales como se indica a continuación (se ha incluido entre las conectivas el simbolo de puntuación "," que puede ser entendido como conectiva causal, conjuntiva o disyuntiva).
La ambiguedad que aparece al utilizar el simbolo de puntuación "," como conectiva se ha intentado eliminar definiendo un orden y una prioridad entre las conectivas de la frase donde aparece la coma.

Las reglas que representan las estructuras tenidas en cuenta vienen agrupadas por su semántica (se denomina o1 a la semántica de <fr-dec>, o2 a la de <fr-declarativa>, o3 a la de <fr-sim> y o4 a la de <fr-comp>).

{o1,o2, o1-> o2}
Semantica asociada a <fr-causal>
{o1,o2, o1 <-> o2}
Semantica asociada a <fr-dobimp>

{o3,o1, conj(o3,o1)}

Semantica asociada a <fr-comp>

{o3,o1, dis(o3,o1)}

Semantica asociada a <fr-comp>

{o3,<c,o5>,o3', c(o3,c(o5,o3'))con cE{conj,dis}}

Semantica asociada a <fr-comp>

{o3, o3}

Semántica asociada a <fr-sim>

{o3, no(o3)}

Semantica de <fr-sim-neg> ::= <gn> <no> <gv>

{o3,o3', conj(no(o3),no(o3'))}

Semantica de <fr-sim-neg> ::= <gn> <no> <gv> <ni> <gv>

La semántica de la frase simple o proposición viene representada por la lista de la semántica general asociada a cada una de las palabras (diferentes de particulas de conexión o articulos) que la forman.

No se ha intentado solucionar el problema de frases simples en castellano con estructura sintáctica diferente y analogo sentido. Puede argumentarse que con frases aisladas (no asi con tratamiento de textos) por muy complejas que sean (conexiones mediante particulas) no suele darse el caso. Es decir no se esperan aunque si sean analizadas frases del tipo:

Si la lluvia cae, las personas utilizan el paraguas cuando llueve"

f1 <- la lluvia cae

f2 <- las personas utilizan el paraguas

f3 <-llueve

[f1 -> (f3 -> f2)]

Siendo f1 = f3.

"llueve y llueve"

[f1 y f1] equivalente a [f1].

Estructura semántica en cálculo de predicados

La semántica en cálculo de predicados se construye ampliando la semántica en lógica proposicional al construir de diferente forma la semántica de las frases simples.
El cálculo de predicados permite expresar relaciones o propiedades sobre objetos individuales o conjuntos.
Para introducir esta nueva caracteristica a la definición semántica anterior se le añaden o modifican las reglas como aparece en el apartado de descripción de la gramática.

Estructura semántica en lógica concepto atributo valor

Disponer de una semántica basada en ternas Concepto, atributo, valor es de gran importancia cuando el interface en castellano es utilizado como comunicación con un sistema experto (con conocimiento expresado en un formalismo similar). De esta forma pueden realizarse la mayor parte de las tareas propias en ese dominio en castellano, es decir introducir, recuperar o modificar conocimiento y preguntar al sistema. Facilita de esta forma la detección de diversos tipos de errores y se consigue una mayor claridad en la comunicación.

Los elementos de una terna pueden ser entendidos de la siguiente forma: Concepto es un ente abstracto o fisico general del que se va a realizar una descripción, atributo es una caracteristica general o un nombre de propiedad de un concepto y valor es un valor del atributo de un concepto.

Un ejemplo:

CONCEPTO	ATRIBUTO	VALORES
Banco	Num-Sucursales	Nro natural
	Ambito	Local, Estatal, Internacional
Sucursal	Num-Empleados	Nro natural
	Localización	Madrid, Zaragoza, Barcelona
Empleados	Rango	Director, Subdirector
	Sector	Bancario, Servicios

Se han realizado dos sistemas con este tipo de lógica modificando del interface general en ambos casos, sólo el tratamiento semántico de las frases simples con respecto al sistema con el que realiza la comunicación.
En el primer interface se modificó la clasificación de algunas palabras comprobandose posteriormente que no hubiera sido necesario. En general la solución utilizada es la de construir la semántica en cálculo de predicados y a continuación transformarla en este tipo de representación con

las hipótesis siguientes:

hip1: Los nombres de relación o propiedades definidas por nombres comunes definen los atributos o los conceptos.
hip2: Los adjetivos, los nombres propios y los comunes sin complementos representan los valores de atributos.

Se describe muy brevemente el primer prototipo de sistema realizado de decisión sobre la humedad del suelo por ser el que mas difiere del sistema Sirena. El modulo de tratamiento del lenguaje debia comunicarse con el prototipo de un sistema experto en situaciones producidas por el aqua (lluvia, rios, etc). Se definieron los atributos nivel, caudal, zona, cuenca, población, instante, humedad y problema. Los atributos se relacionan de la siguiente manera:

nivel <- zona, instante valor-nivel
caudal <- cuenca, instante, valor-caudal
problema <- población, instante, valor-problema
humedad <- zona, instante, valor-humedad

A su vez cada uno de los atributos tiene asociado un rango o conjunto de valores:

zona = {naranjal, carretera, rioalto, ...}
cuenca = {uno, dos, tres}
población = {ciudad, pueblo}
valor-problema = {nulo, parcial, total}
valor-caudal = {torrencia, serio, bajo, nulo}
valor-nivel = {normal, superior, bajo, nulo}
valor-humedad = {alta, media, baja}
instante = {números}

Algunos ejemplos:
"El caudal de la cuenca uno en el instante 2 es bajo"
conj(caudal(uno,2,bajo),cuenca(uno),inst(2))
"¿Cual es el caudal de la cuenca uno?"

<s,conj(caudal(uno,i,s),cuenca(uno),inst(i))>

"Hay un problema parcial en la ciudad"

conj(problema(ciudad,i,parcial),poblacion(ciudad),inst(i))

"¿Que nivel hay en el naranjal en el instante 3?"

<v,conj(nivel(naranjal,3,v),zona(naranjal),inst(3))>

Se exige semánticamente que el valor de un atributo pertenezca a su rango. Sintacticamente (en correspondencia con los tres tipos de semántica) se han clasificado las palabras relevantes en tres tipos (cero, uno o dos). Las del tipo cero son aquellas que representan valores de atributos, su semántica no solo indica el código de ese valor sino tambien a que atributo (o atributos) esta asociado:

<superior,pal-t0,sin.q,nivel(superior)>

Las de tipo uno son los identificadores de los atributos y su semántica indica el codigo en el sistema (que puede ser el mismo):

<zona, pal-t1,sin.fem,zona>

Finalmente las palabras de tipo dos son los atributos con tres argumentos y su semántica es el codigo que los representa en el sistema.

<nivel,pal-t2,sin.mas,nivel(zona(i),inst(j),v)>

Los únicos verbos son el ser y el haber en tercera persona.

Este sistema dispone de una explicación del razonamiento a partir de la traza (arbol Prolog). Se realiza con la traza un preproceso que evitaba informes excesivamente prolijos o repeticiones. Hay una serie de patrones que aplicados a las reglas de conocimiento del sistema daban explicaciones como:

Dado que :
el nivel de la humedad del suelo es alto (dato)
llueve mucho (dato)
se deduce (regla3) que
hay un peligro parcial de acceso a la ciudad.

V.3.2 GRAMATICA DEL CASTELLANO

Como ya se indicó en la introducción no es posible realizar un interface que abarque todo el castellano por lo que se elige un subconjunto de él.
El sistema Sirena admite frases de bastante complejidad pero no deja de ser un subconjunto de todas las posibles.
Su elección se realizó en base a su utilización. El tipo de sistemas con los que se pretende trabajar (sistemas basados en el conocimiento) tienen bien definido el dominio sobre el que versan y puede ser amplio pero no muy elaborado. Es por ello por lo que el tipo de frases de comunicación es sencillo y no aparecen ambiguedades complejas, metáforas y otros giros gramaticales utilizados en literatura pero no en textos "informáticos".
Ademas la representación del conocimiento en estos sistemas con los que se trabaja está en general en forma declarativa, con el formalismo lógico de clausulas de Horn, o en cálculo de predicados.
Siguiendo un curso básico de enseñanza de modelización en cálculo de predicados se aprenderán la mayor parte de las frases que estan implementadas en este trabajo.

La gramática utilizada en Sirena describe tanto las estructuras sintácticas de sus componentes como las semánticas o morfológicas (como se observa en el prototipo de gramática realizado en el apartado uno de este capitulo). En cierta forma la morfologia de las palabras esta determinada por el tipo de estructuras gramaticales aceptadas ya que por ejemplo en la frase "Manuel lleva los zapatos rotos" rotos se considera un adjetivo en lugar de la forma del verbo romper.
La introducción en el sistema de nuevas caracteristicas o restricciones a verificar por las reglas existentes es sencillo de realizar ya que, en general, solo es necesario modificar o añadir algun argumento o bien introducir condiciones en la parte derecha de la regla Prolog correspondiente. Así mismo añadir reglas gramaticales para

definición de otras composiciones de palabras correctas implica una adición de reglas Prolog en el programa de la gramática.
La semántica definida se basa en las hipótesis dadas en el apartado 3.1 de este capítulo.

Entre las caracteristicas gramaticales implementadas se encuentra la coordinación entre frases (frases compuestas) con posibilidad de utilizar el signo "," como conector ambiguo (causa, disyunción o conjunción), y el estudio de la concordancia en las frases (ya comentado en otro apartado).

Se describe a continuación la gramática implementada. Hay dos grandes grupos el de las frases declarativas y el de las frases interrogativas. Las primeras pueden ser causales, compuestas o simples y las segundas pueden estar introducidas por los pronombres quien, cual, cuanto o que, o bien ser frases con simbolos de pregunta (¿, ?) de tipo declarativo o frases simples con el verbo en primer lugar. El grupo nominal puede estar constituido por una frase relativa tercer grupo de frases.

Las reglas que representan a estas estructuras se describen en BNF y su semántica se indica con una secuencia entre llaves siendo a partir de los primeros elementos de la secuencia (aportados por los componentes de la parte derecha de la regla) con los que se construye (por unificación) la semántica de la parte izquierda de la regla.

No se refleja en este apartado las restricciones de concordancia entre los componentes de los distintos grupos por considerarse redundante con su descripción en la implementación de la gramática y relativamente sencilla su lectura en las reglas Prolog que implementan las reglas BNF (como se puede observar en la gramática prototipo donde si aparecen).

Como es de esperar las caracteristicas de la implementación ya han sido expuestas en los diferentes apartados de descripción de los elementos del sistema Sirena. Los programas lógicos describen la relación entre la entrada y la salida, es decir indican "que hay que resolver" (semántica declarativa) y "como resolverlo" (semántica procedural), pero ambas cosas se expresan de idéntica forma siendo el interprete (en este caso) el que ejecuta con estrategia predefinida. De esta forma la distancia entre la interpretación declarativa y la procedural es mínima suponiendo una gran ventaja para el diseño y la comprensión de los programas. (Programar con el lenguaje Prolog es especificar las caracteristicas en terminos de relaciones y propiedades de la solución esperada junto con el control). Luego la descripción de los elementos gramaticales y su implementación es muy parecida. A pesar de ello se explica cual es la estructura Prolog utilizada para los diferentes componentes.

Los ejemplos que aparecen al final de la definición de los constituyentes gramaticales son copia de sesiones Prolog. En cada estudio de frase se muestra la ejecución Prolog procesando con el módulo "Dessin" (del Prolog II) cada estructura asociada (con objeto de hacer mas legibles aquellas que son extensas).

V.3.2.1 DESCRIPCION

Se comienza a continuación la descripción de la gramática a partir de sus componentes.

FRASES DECLARATIVAS

Son frases que expresan conocimiento causal, afirmativo o negativo y que tienen por semántica una formula.

<frase declarativa> ::= <frase causal> | <frase compuesta> | <frase simple>

{o1,o2, imp(o1,o2)}

<frase causal> ::=
<si> <frase dec> <entonces> <frase declarativa> |
<si> <frase dec> <,> <frase declarativa> |
<frase dec> <si> <frase declarativa> |
<frase dec> <solo si> <frase declarativa> |
<frase dec> <suficiente para> <frase declarativa>|
<para> <frase dec> <es suficiente que> <frase declarativa>|
<frase dec> <necesario para> <frase declarativa> |
<para> <frase dec> <es necesario que> <frase declarativa> |
<basta> <frase dec> <para> <frase declarativa> |
<frase dec> <a menos que> <frase declarativa> |

{o1,o2, conj(imp(o1,o2),imp(o2,o1))}

<frase causal> ::=
<frase dec> <si y solo si> <frase declarativa> |
<frase dec> <es suficiente y necesario para> <frase declarativa>

{o, o}

<frase dec> ::= <frase simple> | <frase compuesta>

{o1,o2, conj(o1,o2)}

<frase compuesta> ::= <frase simple> <y> <frase dec> |
<frase simple> <pero> <frase dec> |
<frase simple> <aunque> <frase dec> |

{o1,o2, dis(o1,o2)}

<frase compuesta> ::= <frase simple> <o> <frase dec>

{o1,o2, conj(o1,no(o2))}

<frase compuesta> ::= <frase simple> <ni> <frase simple>

{o1,<c,o2>,o3, c(o1,c(o2,o3)) con cE(conj,dis)}

<frase compuesta> ::= <frase simple> <,> <frase comas> <c>
<frase simple>

{o1, <c,o1> con cE(conj,dis)}

<frase comas> ::= <frase simple>

{o1,<c,o2>, <c,c(o1,o2)> con cE(conj,dis)}

<frase comas> ::= <frase simple> <,> <frase comas>

{o, o}

<frase simple> ::= <frase sim-afir> |
<frase sim-neg>

{{<i,o1>,o},<i,o1>, o}

{o, o}

<frase sim-afir> ::= <grupo nominal> <grupo verbal>

{{<i,no(o1)>,o},<i,o1>, o}

{o, no(o)}

<frase sim-neg> ::= <grupo nominal> <no> <grupo verbal>

{{<i,no(dis(o1,o2))>,o},<i,o1>,<i,o2>, o}

{o1,o2, conj(no(o1),no(o2))}

<frase sim-neg> ::= <grupo nominal> <no> <grupo verbal>
<ni> <grupo verbal>

FRASES INTERROGATIVAS

En la semántica el conector principal no es una conectiva lógica sino un identificador que indica el tipo de respuesta esperada si se procesa con un módulo de evaluación en una base de conocimiento (si comienza por "que" o "quien" se espera la lista de objetos "i" que verifican la formula interior "o").

{{<i,o1>,o},<i,o1>, si-no(o)}

<frase id> ::= <verbo ser> <grupo nominal> <atributo> |
<verbo tener> <grupo nominal> <complemento-t>|

<grupo verbal> <grupo nominal>

{<i,o1>, si-no(o)}

<frase-id> ::= <verbo haber> <complemento-h>

{o, o}

<frase-id> ::= <frase declarativa>

(id: interrogativa declarativa)

{<i,o>, quien(i,o)}

<frase icq> = <pron quien> <verbo 0> |

<pron quien> <grupo verbal>

(icq: interrogativa introducida por el pronombre relativo quien)

{<i,<j,o1>>,{<i,o1>,o}, que(j,o)}

<frase iq> ::= <pron rel> <verbo 1> <grupo nominal> |

{<i,<j,<k,o1>>,{<i,o1>,o}, que(j,o)}

<frase iq> ::= <pron rel> <verbo 2> <grupo nominal> |

{<i,<j,<k,o1>>,{<i,o1>,o}, que(j,o)}

<frase iq> ::= <pron rel> <verbo 2> <grupo nominal>

<grupo nominal> |

{{<i,o1>,o}, <i,o1>, que(j,o)}

<frase iq> ::= <pron rel> <grupo nominal> <grupo verbal>

(iq: interrogativa introducida por el pronombre relativo que)

{<i,n(i)>,<i,o1>, cuantos(i,conj(n(i),o1))}

<frase ic>::= <adv cant> <nom comun> <grupo verbal>

(ic: interrogativa introducida por el adverbio cuanto)

FRASES RELATIVAS

Las frases relativas semánticamente se refieren a un objeto (i) para expresar caracteristicas sobre él. Por esto tiene asociada una formula que será conectada conjuntivamente a la del grupo en que aparece. La formulación {<i,o1>,o} indica que siendo <i,o1> información del grupo que la contiene, o es la correspondiente a la frase relativa.

{<i,o>, o}

<frase relativa> ::= vacia

{<i,o1>, {<i,conj(o1,o)>,o}}

<frase relativa> ::= <pron rel> <grupo verbal>

{<i,<j,o2>>,<j,o1>, {<i,existe(j,conj(o,conj(o2,o1)))>,o}}

<frase relativa> ::= <preposicion> <pron cuyo> <nombre comun>
<grupo verbal>

GRUPO NOMINAL

La semántica asociada a este grupo toma información del tipo <i,o1> de otro constituyente anterior o posterior de la frase (dice o1 sobre el objeto i) y construye la formula o como se indica a continuación.

{<i,o>,o}

<grupo nominal> ::= vacio

{i, {<i,o>,o1} {<i,o1>,o}}

<grupo nominal> ::= <preposicion> <nombre propio>
<frase relativa>

{{<j,o2>,<i,o1>,o},<j,o3>,{<j,o2>,o3}, {<i,o1>,o}}

<grupo nominal> ::= <grupo articulo> <nombre comun>
<frase relativa>

{{<j,o2>,<i,o1>,o3},<j,<k,o2>>,{<k,o3>,o}, {<i,o1>,o}}

<grupo nominal> ::= <grupo articulo> <nombre comun>
<grupo nominal>

{{<j,o2>,<i,o1>,o3},<j,<k,<m,o2>>>,{<k,o3>,o4},{<m,o4>,o},
{<i,o1>,o}}

<grupo nominal> ::= <grupo articulo> <nombre comun>
<grupo nominal> <grupo nominal>

{{<j,o2>,<i,o1>,o},<j,o3>,{<j,o3>,o4},{<j,o2>,o4},
{<i,o1>,o}}

<grupo nominal> ::= <grupo articulo> <nombre comun>
<adjetivo> <frase relativa>

{{<j,o4>,<i,o1>,o3},<j,<k,o2>>,{<j,o2>,o4},{<k,o3>,o},
{<i,o1>,o}}

<grupo nominal> ::= <grupo articulo> <nombre comun>
<adjetivo> <grupo nominal>

GRUPO VERBAL

La semántica es una estructura <i,o> donde i es un objeto definido por el sujeto de la frase sobre el que expresa o el grupo verbal. No se han tenido en cuenta posibles complementos circunstanciales.

{<i,o>, <i,o>}
<grupo verbal> ::=
<verbo ser> <atributo-s> |
<verbo tener> <complemento-t> |
<verbo estar> <atributo-e> |
<verbo haber> <complemento-h>
{p,{<p,p>,o2},{<p,o2>,o}, <i,o>}
<grupo verbal> ::=
<verbo proposicional> <adjetivo> <grupo nominal>
Verbo sin sujeto (llueve)
{<i,o>, <i,o>}
<grupo verbal> ::= <verbo 0>
Verbo con sujeto (viene)
{<i,<j,o1>>,{<j,o1>,o}, <i,o>}
<grupo verbal> ::= <verbo 1> <grupo nominal>
Verbo con sujeto y complemento directo (estudia)
{<i,<j,<k,o1>>>,{<j,o1>,o2},{<k,o2>,o}, <i,o>}
<grupo verbal> ::= <verbo 2> <grupo nominal> <grupo nominal>
Verbo con sujeto, complemento directo e indirecto (da)

COMPLEMENTO

El complemento acompaña a los verbos tener y haber. Ambos tienen semántica especial ya que no aportan información por si solos.

{n, <i,<j,o1>>, {<j,o1>,o2>},
<i,existe(j,conj(o2,cant(j,n)))>}
<complemento-t> ::= <cardinal> <nombre comun> <adjetivo>
{<i,<j,o1>>, {<j,o1>,o2}, <i,existe(j,o2)>}

<complemento-t> ::= <nombre comun> <adjetivo> |
{<i,<j,<k,o1>>>, {<j,o1>,o2}, {<k,o2>,o}, <i,existe(j,o)>}
<complemento-t> ::= <nombre comun> <adjetivo> <grupo nominal>

{n, <j,o1>, {<j,o1>,o2}, <i,existe(j,conj(o2,cant(j,n)))>}
<complemento-h> ::= <cardinal> <nombre comun> <adjetivo>
{<j,o1>, {<j,o1>,o2}, <i,existe(j,o2)>}
<complemento-h> ::= <nombre comun> <adjetivo> |
{<j,<k,o1>>, {<j,o1>,o2}, {<k,o2>,o}, <i,existe(j,o)>}
<complemento-h> ::= <nombre comun> <adjetivo> <grupo nominal>
{{<j,vacio>,o}, <i,o>}
<complemento-h> ::= <grupo nominal>

ATRIBUTO

Este grupo acompaña a los verbos ser y estar (atributo-s y atributo-e respectivamente).

{{<i,o>,o}, <i,o>}
<atributo-s> ::= <adjetivo>
{<i,o1>,{<i,o1>,o}, <i,o>}
<atributo-s> ::= <articulo> <nombre comun> <adjetivo>
{<i,<j,o1>>,{<i,o1>,o2},{<j,o2>,o}, <i,o>}
<atributo-s> ::= <articulo> <nombre comun> <adjetivo>
<grupo nominal>
{<i,o1>,{<i,o>,o1}, <i,o>}
<atributo-s> ::= <articulo> <nombre comun> <fr relativa>

{{<i,o>,o}, <i,o>}
<atributo-e> ::= <adjetivo>
{{<i,<j,o1>>,o1},{<j,o1>,o}, <i,o>}
<atributo-e> ::= <adjetivo> <grupo nominal>

GRUPO ARTICULO

Su semántica se construye a partir de dos formulas sobre objetos (por ejemplo, la que proviene del sujeto de la frase

y la del grupo verbal). El cuantificador lo define el artículo del grupo (si no lo hay se ha de de construir la estructura del grupo al que pertenece).

{<i,o1>,<i,o2>, existe(i,conj(o1,o2))}
<grupo articulo> ::= <preposicion> <articulo>
{<i,o1>,<i,o2>, existe(i,conj(o1,o2))}
<grupo articulo> ::= <grupo articulo contracto>
{<i,o1>,<i,o2>, conj(o1,o2)}
<grupo articulo> ::= <de> | vacio

ARTICULO

El artículo es el que define la cuantificación de la formula asociada a la frase o del grupo al que pertenece. Se ha elegido por convenio que el artículo definido plural este asociado al cuantificador universal.

<articulo> ::= <art indefinido> | <art definido>
{<i,o1>,<i,o2>, existe(i,conj(o1,o2))}
<art indefinido> ::= un | unos | una | unas
<art definido> ::= el | la
{<i,o1>,<i,o2>, todo(i,dis(no(o1),o2))}
<art definido> ::= los | las

PREPOSICION

No aportan información semántica.
<preposicion> ::= vacia | de | en | para | a

VERBOS

Los verbos haber, tener, ser o estar no aportan información semántica. Son los cuatro verbos que pertenecen al diccionario básico (aparecen en cualquier interface realizado con el sistema). El resto de los verbos se ha categorizado de la siguiente forma:

verbos proposicionales o sin sujeto (ver-prop):

La semántica es un simbolo de predicado 0-ario. Forman por si solos una frase, aunque puden ser complementados. Su semántica es una proposición. Ejemplo: llueve.

verbos de tipo 0 (verbo 0):

Son los verbos intransitivos. La semántica es un predicado unario <ver,i> siendo i el objeto referido en el sujeto. Ejemplo: viene

verbos de tipo 1 (verbo 1):

Son los que aceptan complemento directo. Su semántica es un predicado binario <ver,i,j> siendo i el objeto referido en el sujeto y j el del complemento directo. Ejemplo: estudia

verbos de tipo 2 (verbo 2):

Son los que aceptan complemento directo e indirecto. Su semántica es un predicado ternario <ver,i,j,k> siendo i el objeto referido en el sujeto, j en el complemento directo y k en el complemento indirecto. Ejemplo: da

Solo se admite los verbos en tercera persona singular y plural del presente de indicativo.
La no aparición en la frase de alguno de los posibles complementos del verbo hace que el argumento correspondiente del predicado del verbo se quede sin valor: "Manuel estudia" semánticamente es estudia(Manuel,x).

NOMBRE COMUN

Introduce semánticamente un simbolo de relación. El número de argumentos vendrá determinado por los requerimientos de la frase. Es decir cada uno de los argumentos los produce la existencia de un grupo nominal.
Por ejemplo la frase:

"el hombre viene"

tiene por semántica :

existe(x,conj(hombre(x),viene(x)))

y la frase:

"el hombre de la casa viene"

tiene por semántica:

existe(x,conj(casa(x),
existe(y,conj(hombre(y,x),viene(y))))).

ADJETIVO

Un adjetivo acompaña al nombre o forma parte de un atributo caracterizando en el primer caso al objeto del grupo nominal o en el segundo caso al objeto sujeto de la frase.Por ejemplo "Manuel es feliz" toma por semántica feliz(Manuel). Sólo en el caso de adjetivos como "contento" se admite una formalización binaria:

"Manuel esta contento del juguete"

con semántica:

existe(x,conj(juguete(x),contento(Manuel,x))).

NOMBRE PROPIO

Representa semánticamente a un objeto (i) del dominio de la base de conocimiento.

CARDINALES

Significan un número determinado (cero, uno, dos, tres, cuatro, cinco, seis, siete, ocho, nueve) e introducen el predicado binario "cant(x,n)" que indica que hay un número n de objetos x (es un predicado no lógico que será tratado como tal por el mecanismo de inferencia). En la frase:

"hay dos hombres"

se tiene:

existe(x,conj(hombre(x),cant(x,2))).

La regla de la gramática es:

<cardinal> ::= un | uno | una | dos | tres | cuatro | cinco |
seis | siete | nueve | ocho | cero

ADVERBIOS

El adverbio cuanto/a/os/as introduce el predicado binario "cuantos(x,f)" que significa ¿cuantos objetos x verifican la propiedad f?. Este pronombre introduce el tipo de frases interrogativas categorizadas como "frase-cant", siendo respondidas por el número que verifica la condición en ella expresada. Por ejemplo:

¿Cuantos alumnos miran a la pizarra?

es una frase con semántica:

cuantos(x,conj(alumno(x),
existe(y,conj(pizarra(y),mira(x,y)))))

La regla de la gramática es:

<adv cant> ::= cuanto | cuanta | cuantos | cuantas

Los adverbios si, entonces, necesario, suficiente, solo, menos son comentados en el apartado de Conectores.

PRONOMBRES

El pronombre cuyo/a/os/as y el relativo que introducen a frases relativas y no aportan información semántica. El pronombre que puede aparecer ademas como parte de conector o como símbolo de pregunta introduciendo en este caso la semántica de frase "que(i,o)" que expresa la petición de objetos i que verifican la formula o.

<pron rel> ::= que
<pron cuyo> ::= cuyo | cuya | cuyos | cuyas

Los pronombres quien/es introducen un conector de frases interrogativas que esperan por respuesta (si es evaluada la

formula correspondiente) la lista de objetos que verifican lo expresado en la frase. La semántica asociada quien(i,o) donde i es el objeto (variable) y o la formula que expresa los requerimientos de i.

<pron quien> ::= quien | quienes

CONECTORES

La información semántica que aportan es la de una conectiva lógica.

{dis(no(x),y)}

<conectores> ::= si | entonces | a menos que | solo si | basta | para | necesario| suficiente

{dis(x,y)}

<conectores> ::= o | ,

{conj(x,y)}

<conectores> ::= y | ,

{no(x)}

<conectores> ::= no | ni

V.3.2.2 IMPLEMENTACION

La gramática esta formada por el conjunto de reglas que describen las agrupaciones de palabras que son frases pertenecientes al lenguaje descrito por la gramática.
Cada regla define un grupo de palabras válido o bien la categoria de palabras del vocabulario. La definición de las palabras aparece en la parte diccionario.
Para cada regla de la gramática hay, en general, una regla Prolog. La parte izquierda define la forma de las estructuras asociadas a ese grupo y la parte derecha la de sus componentes.

Para cada categoria hay un predicado con tres tipos de argumentos: el que define lo relativo a la concordancia, el que define la estructura sintáctica y el de la estructura semántica.

```
grupo(<sec-conc,sec(est-sint)>,sec-sem)
sec-conc == <t,1>,<n,<t,cons(1)>>
```

Siendo "grupo" el nombre del constituyente, "t" la traza de genero, numero y a veces otras condiciones relativas al contexto, "n" el nombre de la estructura sintáctica del grupo en definición, "cons(1)" la forma de construcción de la lista "1" asociada al grupo y que indica los posibles errores de concordancia aparecidos en los componentes del grupo, "sec(est-sint)" es la secuencia formada con las estructuras sintácticas asociadas a los componentes del grupo (puede ser vacia), y "sec-sem" la secuencia de formulas de los componentes con las que se construirá la formula semántica final asociada (puede ser vacia). Por ejemplo:

```
frase-declarativa(<<t,1>,est-sint>,est-sem) ->
   ind-causal(est-sint)
   frase-dec(<<t',11>,est-sint>,est-sem)
   ind-causal(est-sint)
```

```
frase-dec(<<t'',12>,est-sint>,est-sem);
```

Luego en una frase declarativa causal cada una de las frases componentes tiene su genero y número independiente (l1, l2, l3 son las listas de posibles errores en cada componente y l la de resultado).

Algunos ejemplos de reglas de la gramática son:

```
frase-dec(<<t,l>,<r,<t,l.<l1>.<l2>>,x,y>>,o) ->
     frase-sim(<<t',l1>,x>,o1)
     resto-dec(<<t'',l2>,r,y>,o1,o);
grupo-verbal(<<t,l>,gv(<t,l.l1.l2>,x,y)>,<i,o>) ->
     verbo(<<ser.t,l1>,x>,n)
     atri-s(<<t,l2>,y>,<i,o>);
nombre-propio(<<t,l>,np(<t,l>,<t',<x>>)>,y) ->
     <x,np,t',y>;
pron-rel(<pron-rel(<x>)>) ->
     <x,pron-rel,t,s>;
```

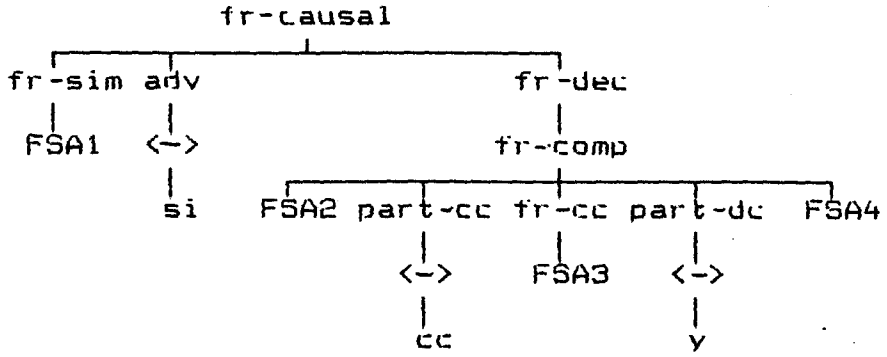
IV.3.2.3 EJEMPLOS

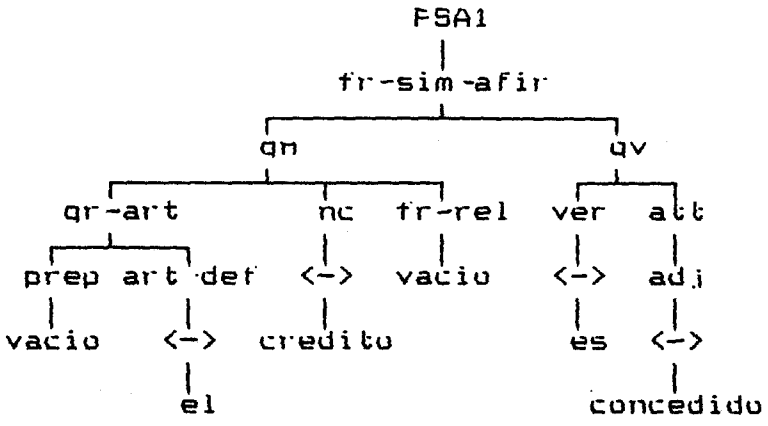
>dib-frase;

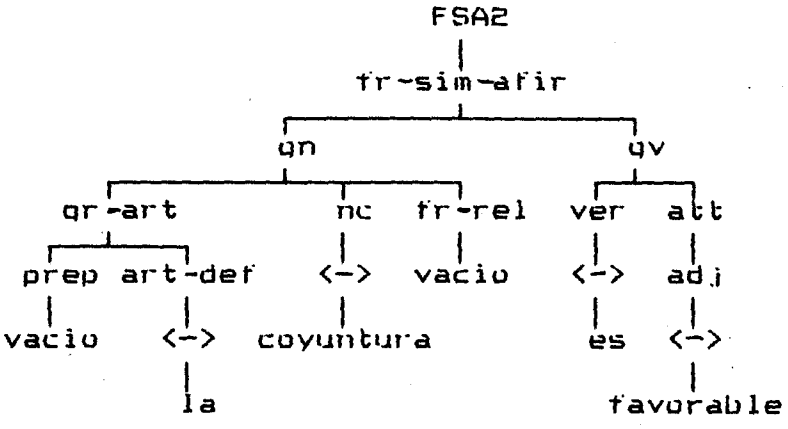
FRASE DECLARATIVA:

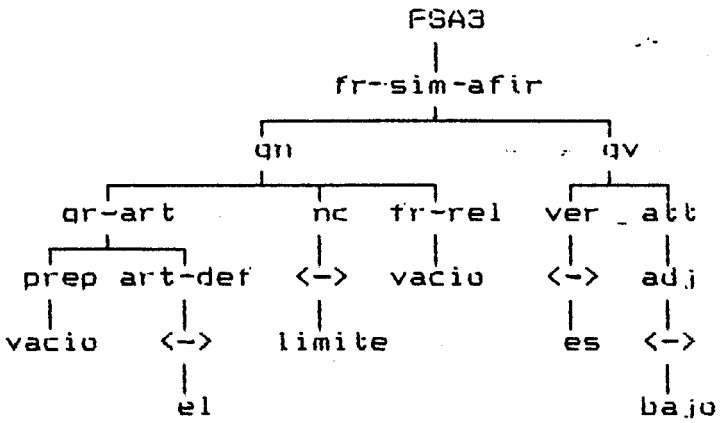
el credito es concedido si la coyuntura es favorable, el limite es bajo y el plazo es corto.

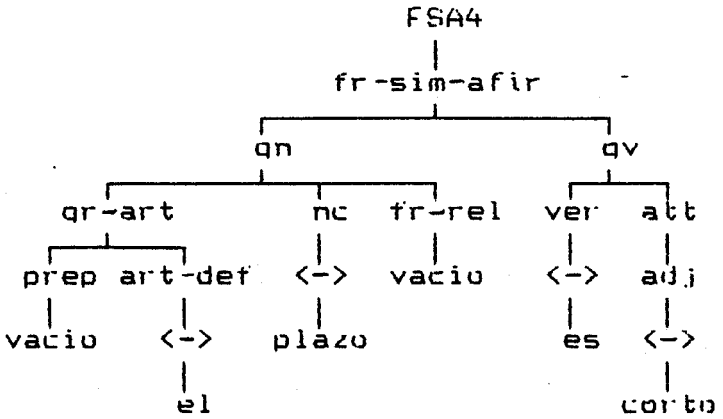
SINTAXIS:



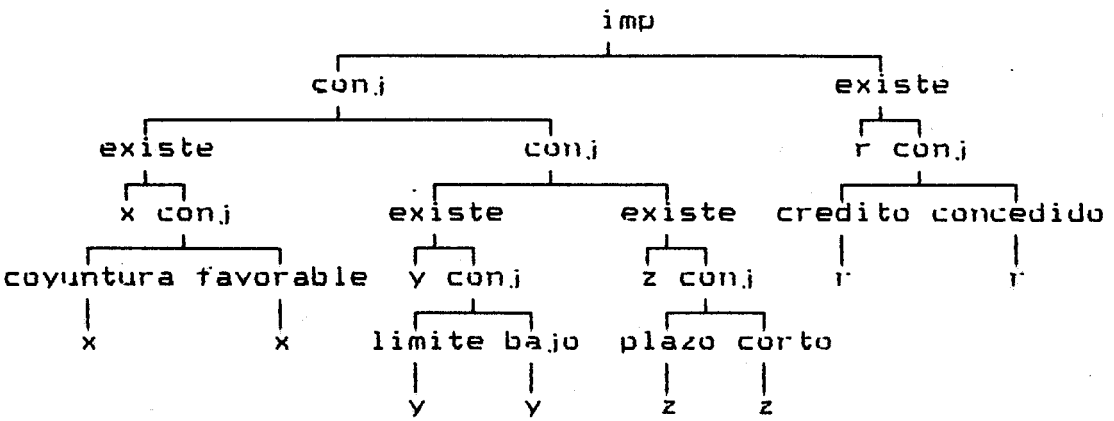





FSA3
fr-sim-afir
gn
gv
gr-art
nc
fr-rel
ver
att
prep
art-def
<->
vacio
<->
adj
vacio
<->
limite
es
<->
el
bajo

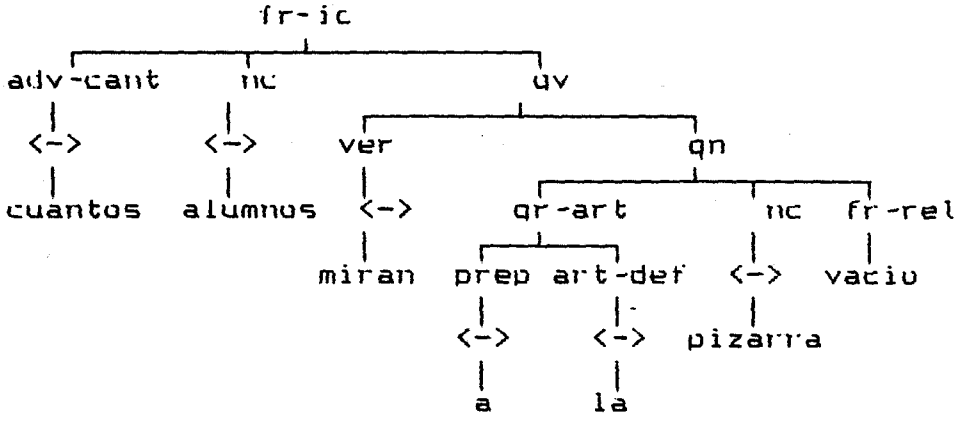

FSA4
fr-sim-afir
gn
gv
gr-art
nc
fr-rel
ver
att
prep
art-def
<->
vacio
<->
adj
vacio
<->
plazo
es
<->
el
corto

SEMANTICA:

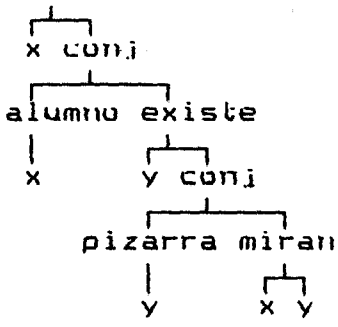

imp
conj
existe
existe
conj
r conj
x conj
existe
existe
credito concedido
coyuntura favorable
y conj
z conj
r
r
x
x
limite bajo
plazo corto
y
y
z
z

()
>

```
>dib-frase;

FRASE INTERROGATIVA:

¿cuantos alumnos miran a la pizarra?

SINTAXIS:
```




```
SEMANTICA:

cuantos
```

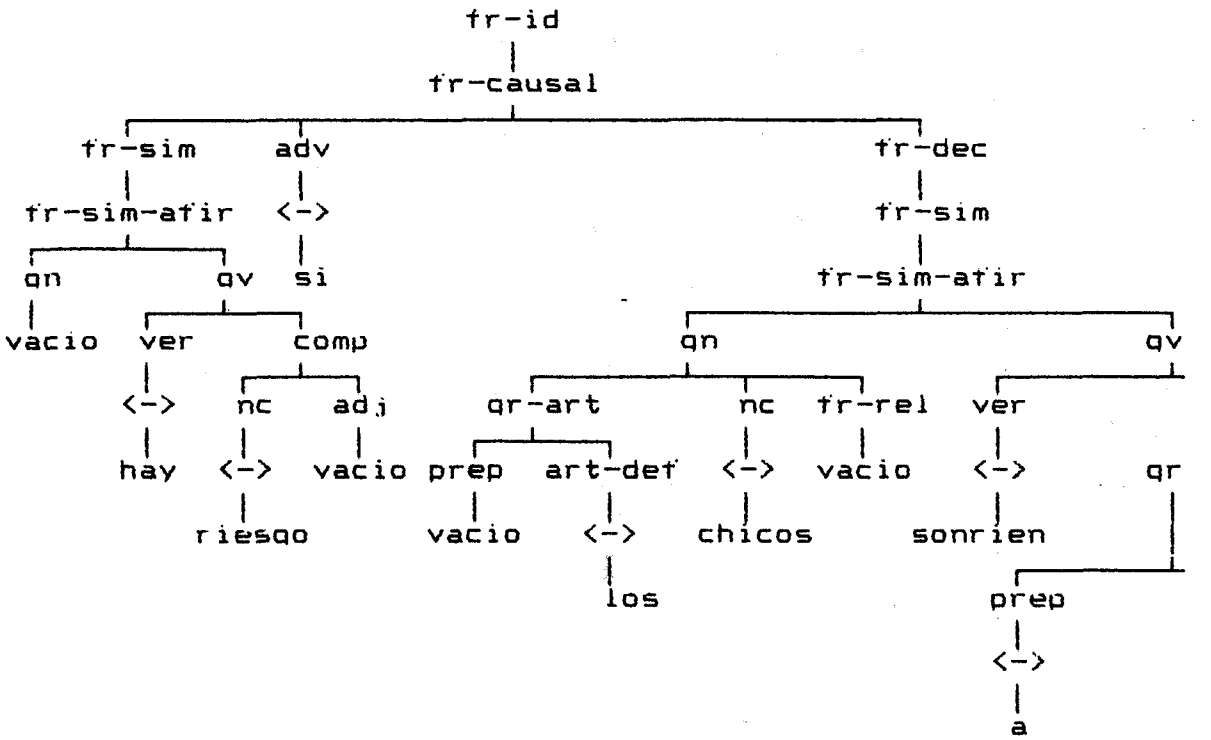



```
()
```

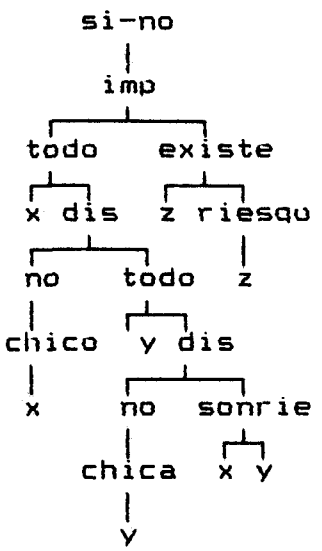
FRASE INTERROGATIVA:

¿hay riesgo si los chicos sonrien a las chicas?

SINTAXIS:



SEMANTICA:

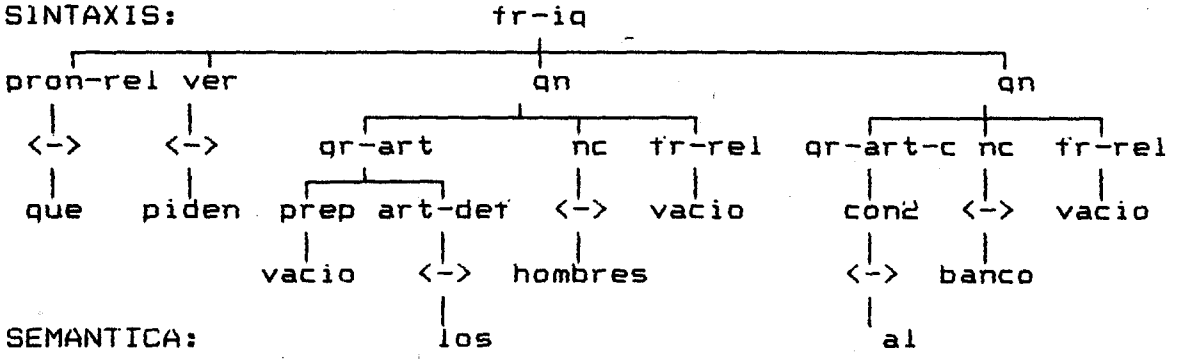


{}

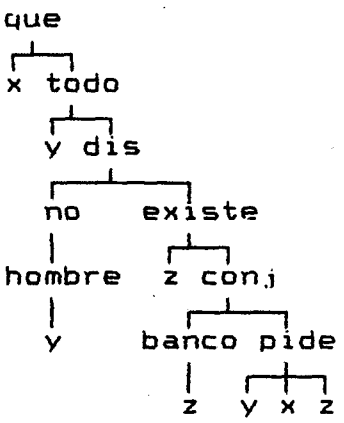
FRASE INTERROGATIVA:

¿que piden los hombres al banco?

SINTAXIS:



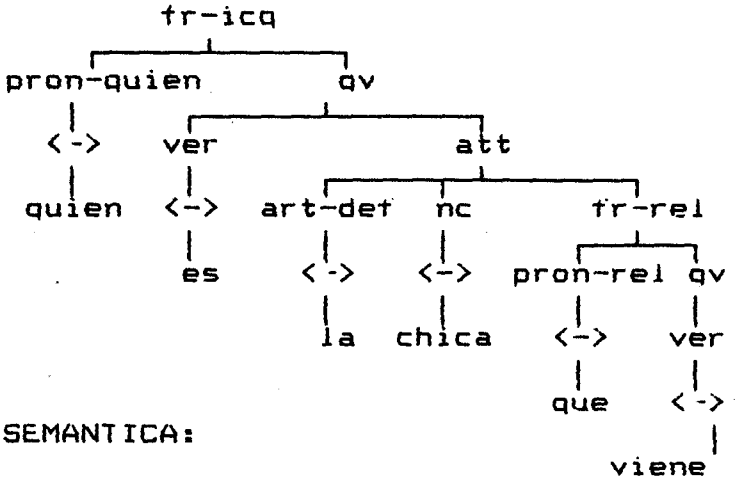
SEMANTICA:



FRASE INTERROGATIVA:

¿quien es la chica que viene?

SINTAXIS:



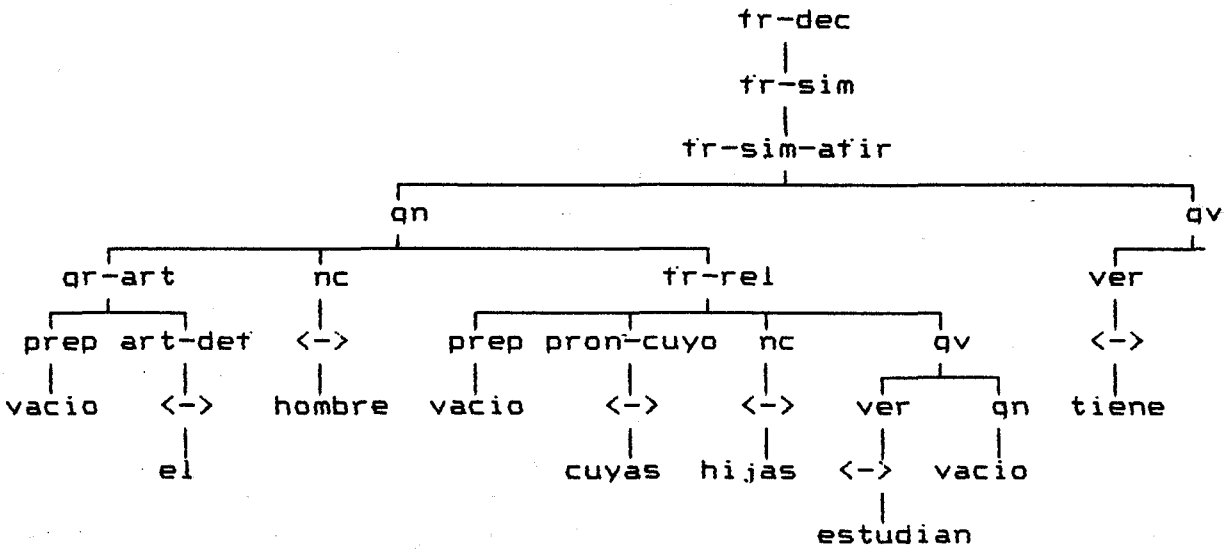
SEMANTICA:

>dib-frase;

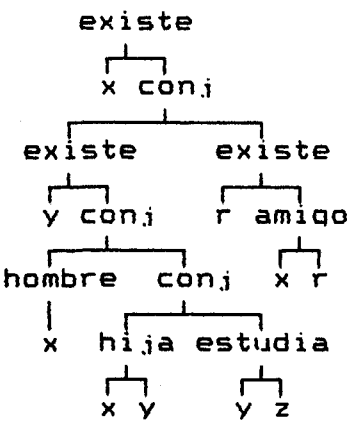
FRASE DECLARATIVA:

el hombre cuyas hijas estudian tiene amigos.

SINTAXIS:



SEMANTICA:

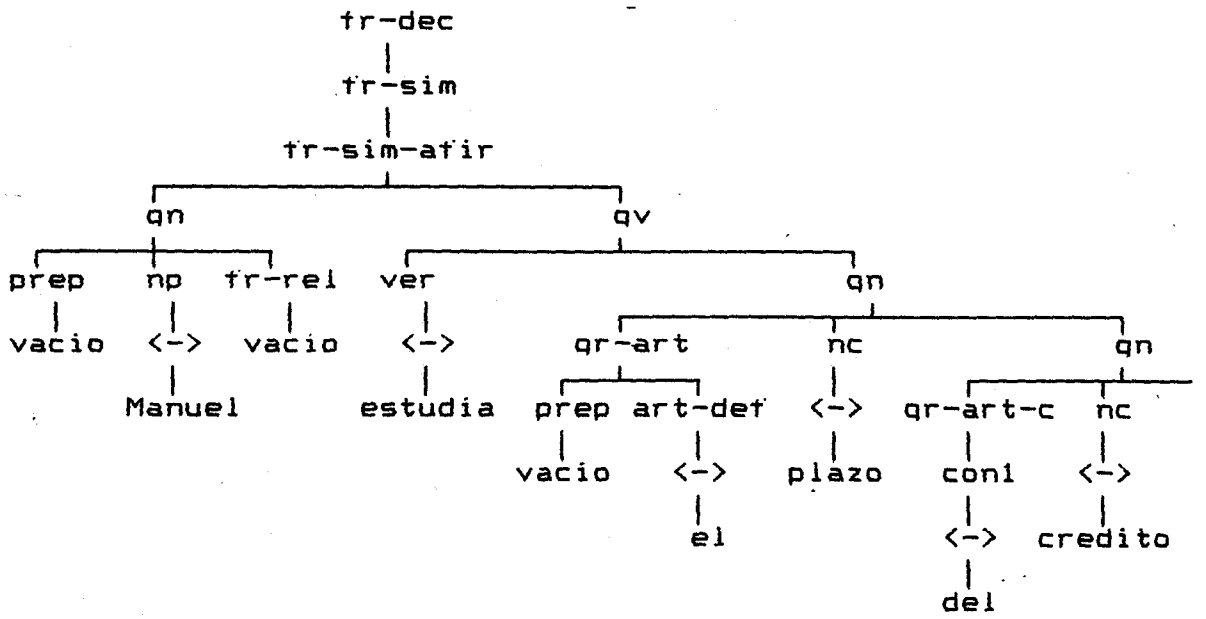


()

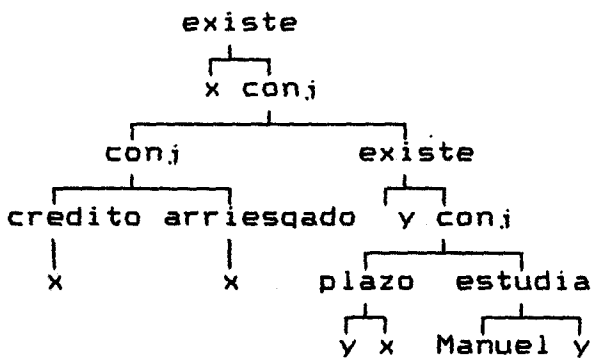
FRASE DECLARATIVA:

Manuel estudia el plazo del credito que es arriesgado.

SINTAXIS:



SEMANTICA:



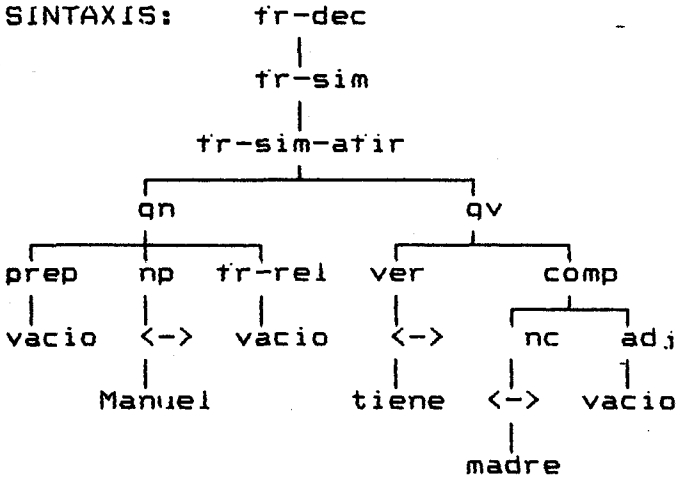
()
>

>dib-frase;

FRASE DECLARATIVA:

Manuel tiene madre.

SINTAXIS:

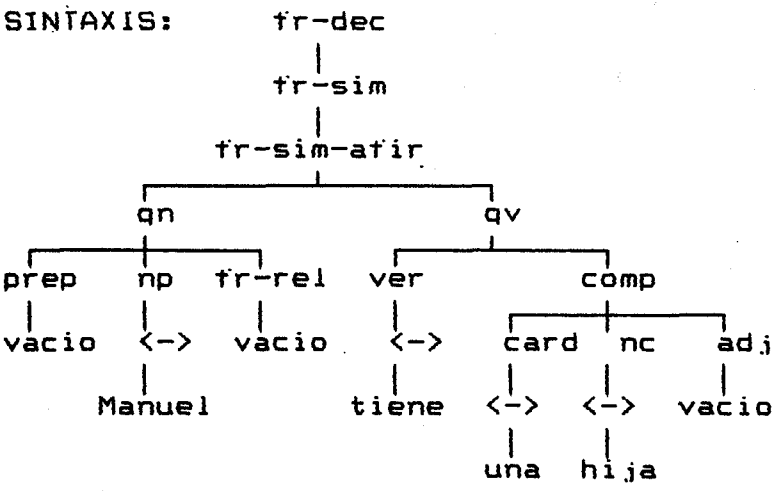


SEMANTICA:

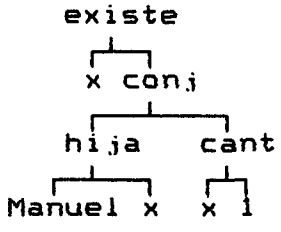
FRASE DECLARATIVA:

Manuel tiene una hija.

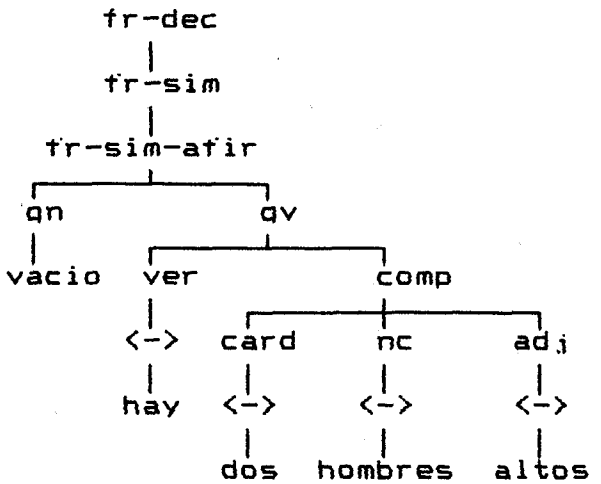
SINTAXIS:



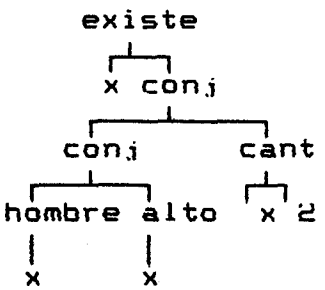
SEMANTICA:



```
>dib-frase;

FRASE DECLARATIVA:

hay dos hombres altos.

SINTAXIS:
```




```
SEMANTICA:
```

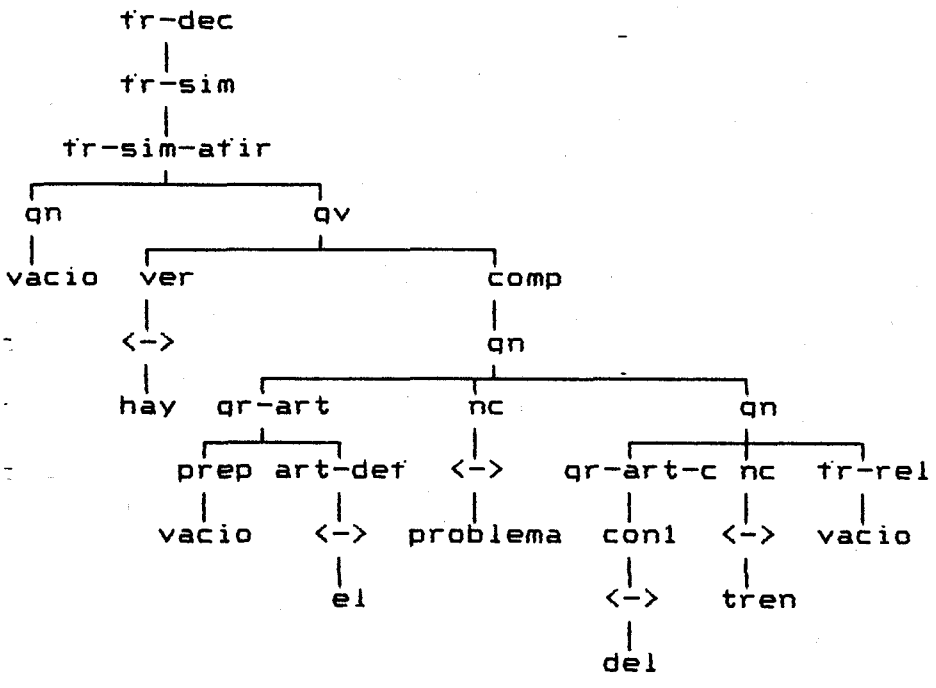



```
()
```

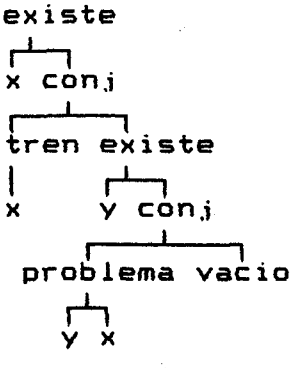
FRASE DECLARATIVA:

hay el problema del tren.

SINTAXIS:



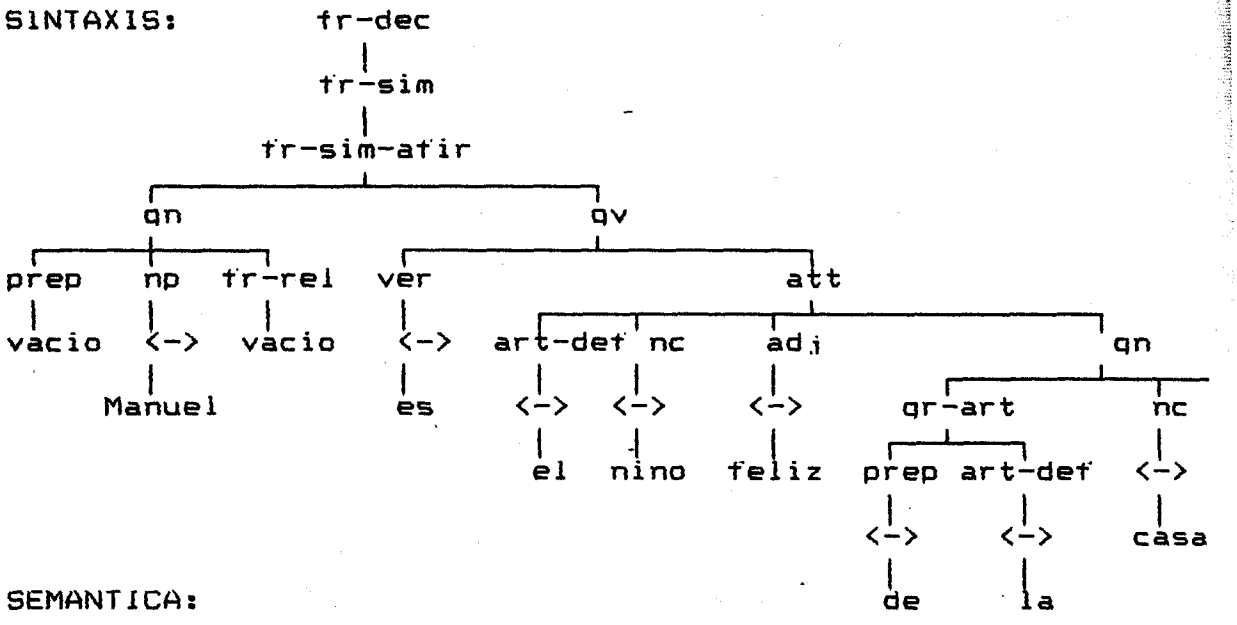
SEMANTICA:



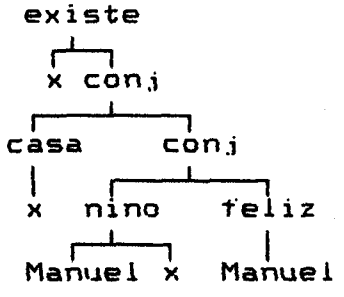
FRASE DECLARATIVA:

Manuel es el nino feliz de la casa.

SINTAXIS:



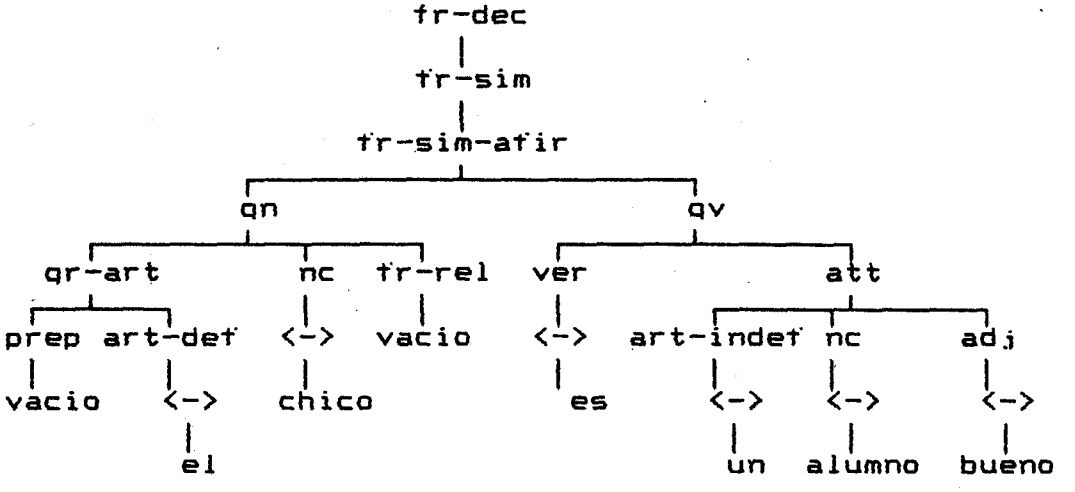
SEMANTICA:



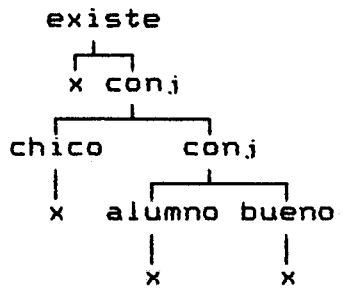
FRASE DECLARATIVA:

el chico es un alumno bueno.

SINTAXIS:



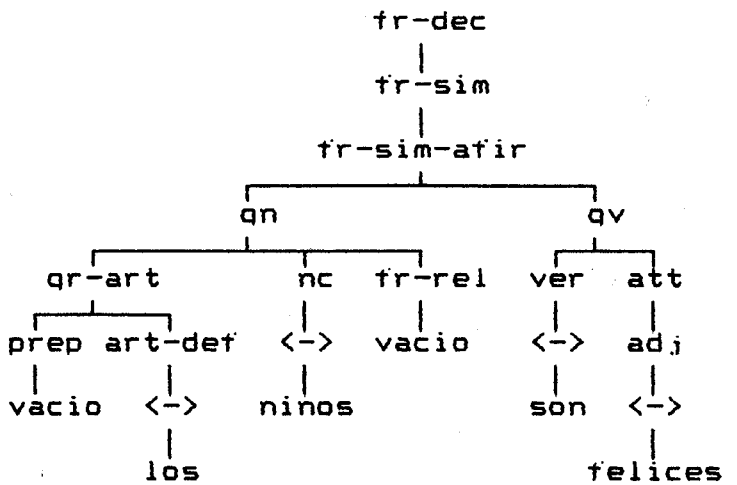
SEMANTICA:



FRASE DECLARATIVA:

los ninos son felices.

SINTAXIS:

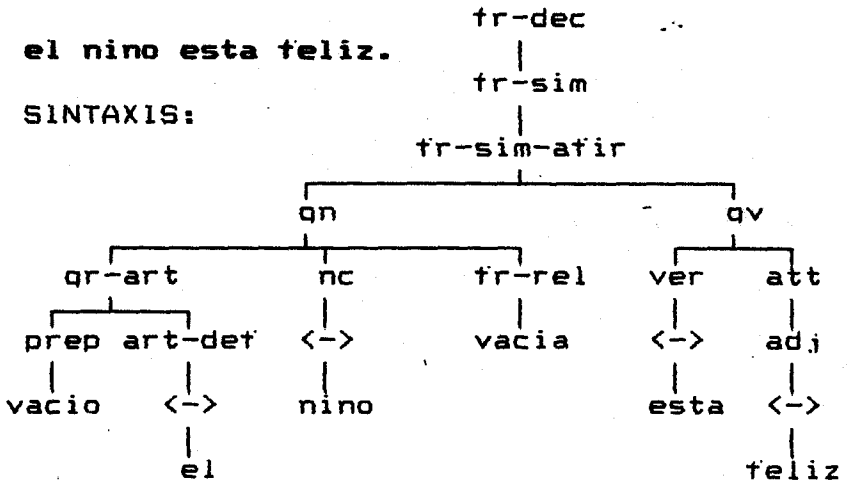


SEMANTICA:

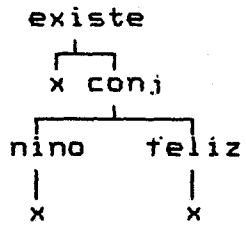
FRASE DECLARATIVA:

el nino esta feliz.

SINTAXIS:



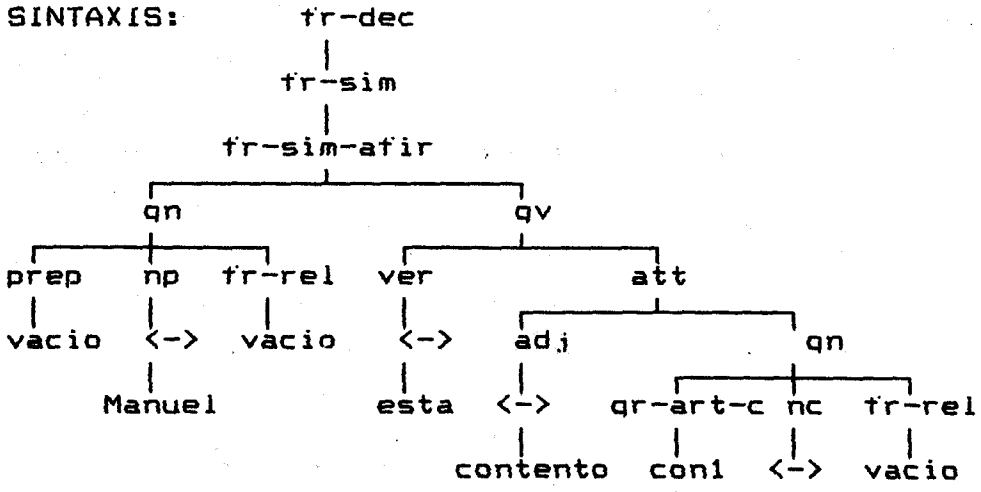
SEMANTICA:



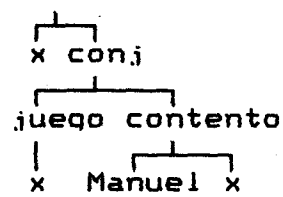
FRASE DECLARATIVA:

Manuel esta contento del juego.

SINTAXIS:



SEMANTICA:



V.3.3 DICCIONARIO

El diccionario de la gramática contiene definiciones de palabras del sistema. Una parte esta formada por las definiciones de palabras "básicas", es decir pueden aparecer en cualquier dominio de aplicación. Son palabras de las siguientes categorias: articulos (gr-art-c, art-def, art-indef), pronombres (pron-rel, pron-cuyo, pron-quien, pron-cant), preposiciones (prep), conectores, numerales, verbo-ser, verbo-tener, verbo-estar y verbo-haber.

Para cada palabra se dispone de la siguiente información almacenada en una regla Prolog con la parte izda formada por la n-tupla:

<palabra,cat-sintáctica,numero.genero,semántica>

Ademas las palabras estan agrupadas por su categoria sintáctica, teniendose para cada raiz de palabra una regla Prolog cuya parte izada esta formada por un predicado unario que indica la categoria sintáctica y un argumento, lista de n-tuplas <numero.genero,palabra>. De esta forma se tiene posibilidad de acceder a todas las palabras con igual raiz (no es posible hacerlo con la estructura del resto del diccionario debido a la implementación).

El acceso al diccionario para recuperar información puede hacerse con sentencias Prolog propias al tipo de regla del diccionario o, con los predicados palabras(c) | c es una categoria sintáctica o con palabra(x) | secuencia de letras candidata a ser palabra del diccionario.

V.3.4 GENERACION DE EXPLICACIONES

Al ser un sistema de comunicación con un sistema basado en el conocimiento hay otra tarea importante a realizar en castellano y es la explicación del razonamiento sequido a partir de la prequnta y hasta la solución aportada.

Como ya se ha indicado en los sistemas realizados se ha construido el módulo de explicación con metodología diferente a la utilizada para la adquisición o comunicación con el sistema.

En general se quarda una traza del razonamiento sequido hasta llegar a la solución. Esta traza es la que determina el tipo de construcción de la explicación. Si los razonamientos son pequeños es decir intervienen pocas reglas, pocos datos entonces sólo suele haber un preproceso de la traza en el que se ordenan los elementos a explicar y se quitan explicaciones repetidas. Si por el contrario el razonamiento es extenso, se opta por ocultar partes del razonamiento.
La explicación se construye con patrones que se rellenan a partir de la traza disponible.
El tipo de implementación de la traza varia según el sistema realizado. En realidad es el tamaño de un razonamiento estandar el que determina su modo de almacenamiento. Se ha observado que si este es corto se quarda en un arbol Prolog y si es extenso en un conjunto de reglas Prolog como el del ejemplo que se presenta en este apartado.

En el ejemplo que se muestra a continuación sòlo se explican los "niveles superiores" del razonamiento para realizar un informe final con objeto de eliminar información poco relevante y hacer mas legible el informe (aunque si eran solicitados otras partes del razonamiento se explicaban tambien, es decir se tenian distintos niveles de explicación). La traza se almacenaba en reglas Prolog indicando los valores consequidos para cada atributo con su

valor de certeza. Es un proceso con el sistema experto "Director ideal" [Cuena,García-Serrano, 87].
El sistema aconseja sobre la conveniencia de conceder créditos con un límite determinado (cantidad a prestar), decidiendo a partir de ciertas cuenta caracteristicas del demandante, de los avales presentados y de la situación coyuntural del banco. Se contruyeron en cuenta distintos motores de inferencia mostrandose a continuación el informe de un "caso" procesado con el motor de inferencia de tipo Mycin. (Se muestra tal y como lo presenta una sesión Proloq con el sistema).

RESULTADOS MYCIN DEL CASO 04

La RESOLUCION FINAL es no conceder por desconfianza y por razones tecnicas.
La DECISION INICIAL es favorable con grado de certeza 0.11 y desfavorable con grado de certeza 0.84.
La MALA FE es notoria con grado de certeza 0.64 y hay indicios con grado de certeza 0.23.

Es un caso en el que :
si hay papel
no hay avales

Entre los calculos intermedios se destaca que:

La SOLIDEZ EFECTIVA DEL CLIENTE es equilibrada con grado de certeza 1.00.
La LIQUIDEZ DEL CLIENTE es estricta con grado de certeza 0.50 y es mala con grado de certeza 0.50.
El RIESGO ECONOMICO es normal con grado de certeza 1.00.
La CAPACIDAD DE REEMBOLSO es insuficiente con grado de certeza 0.62 y es suficiente con grado de certeza 0.25 .
La GARANTIA es insuficiente con grado de certeza 0.35 y es suficiente con grado de certeza 0.44 .
El VALOR DE LAS COMPENSACIONES POR SALDO es nula con grado de

certeza 1.00.

El VALOR DE LAS COMPENSACIONES POR PAPEL es media con grado de certeza 0.25 y es baja con grado de certeza 0.75.

El VALOR DE LAS COMPENSACIONES es baja con grado de certeza 0.75.

La CALIDAD DEL NEGOCIO es mala con grado de certeza 0.20.

La IMAGEN DEL CLIENTE es negativa con grado de certeza 0.75.

La VALORACION DEL CLIENTE es buena con grado de certeza 0.37 y es mala con grado de certeza 0.50.

La RENTABILIDAD es insuficiente con grado de certeza 0.50 es suficiente con grado de certeza 0.50.

El NIVEL DE FRAUDE ESTRUCTURAL es alto con grado de certeza 0.64 y es medio con grado de certeza 0.23.

El NIVEL DE FRAUDE DE PAPEL es nulo con grado de certeza 1.00.

La FIABILIDAD es alta con grado de certeza 1.00.

Los INDICIOS NEGATIVOS son nulos con grado de certeza 1.00.

El NIVEL DE FRAUDE DE LA OPERACION es nulo con grado de certeza 1.00.

V.4 TAREAS LINGUISTICAS QUE REALIZA EL SISTEMA

Las tareas del sistema son :

1. ANALISIS DE UNA FRASE
2. INTRODUCIR PALABRAS EN EL DICCIONARIO
3. COMUNICACION CON LA BASE DE CONOCIMIENTO

   Introducir nuevo conocimiento
   o Confirmación de prequnta
   o Salida de valores para los que la prequnta es cierta.

Al acabar cualquiera de las tres tareas el usuario vuelve a ser prequntado por la siquiente a realizar. En el menú de selección de tarea ademas se puede optar por una cuarta opción que es TERMINAR la sesión.

Hay varios puntos del proceso del sistema en los que el usuario ha de elegir una opción entre varias presentadas en un menú. La respuesta dada ha de ser siempre un número entero o un identificador (dentro del ámbito de los propuestos) pero la respuesta con una sola letra puede provocar problemas irresolubles en la dirección realizada por el sistema sobre el proceso (debido a la versión del lenquaje utilizado).

La tarea de explicación del razonamiento es particular a la aplicación por lo que no se trata en este apartado.

V.4.1 ANALISIS DE UNA FRASE

Esta tarea de proceso de una frase consta de los siquientes pasos:

Lectura de la frase.
Test de reconocimiento de las palabras de la frase.
Analisis sintáctico semántico.
Detección-recuperación de errores de concordancia.
Salida de estructuras asociadas a la frase

A continuación se describe cada una de ellas:

Lectura de la frase

Este módulo realiza la lectura de una frase y detecta si la frase es declarativa o interroqativa.
El sistema maneja la frase como una lista de identificadores Proloq. Para ello es necesario procesar la lista de entrada ya que el único predicado predefinido que puede ser utilizado de lectura toma la frase como una lista de palabras entre comillas. Estas son eliminadas en todas las palabras con más de una letra. Si son palabras con una letra y pertenecen al conjunto (a, o, y) son duplicadas y en otro caso permanecen con comillas (se recuerda que una letra minuscula en el Proloq con el que se ha implementado denota variable).

Por las caracteristicas del lenguaje de implementación (las letras mayusculas son distintas de las minusculas) se utiliza el convenio de que las palabras han de ser escritas con minuscula salvo los nombres propios (por ser considerados siempre comenzando con mayuscula), sin acentos y sin utilizar la letra "ñ" (por ser caracteres dobles no reconocidos).

Una frase es una secuencia de palabras acabada en "." con lo que la frase es estudiada como declarativa o bien es una secuencia de palabras que empieza por el simbolo "¿" y

acaba por "?" con lo que la frase es estudiada como interrogativa.
Por error de escritura si la frase "solo" acaba con el simbolo "?" se considera interrogativa pero si comienza con "¿" y acaba con el simbolo "." se pregunta al usuario que tipo de frase desea sea analizada.
La implementación de la introducción de la frase se realiza con un predicado predefinido que acepta secuencias de palabras acabadas en {".", "?", "!"}. El tercer tipo de simbolo no se ha creido conveniente utilizar de momento en este sistema por razones pragmáticas.

Para depuración y prueba del sistema fundamentalmente se ha introducido la posibilidad de introducir una secuencia de frases (independientes entre si, es decir no son entendidas como texto) separadas por los simbolos de puntuación "." o "?" aceptados como terminación de introducción de frase que acaba con "fin.". El resultado del proceso de cada una de ellas queda almacenado en el fichero "prolog.log".

Ejemplo de introducción de frases:

```
>com-op;
SELECCIONE UNA OPCION :
1: ANALISIS DE UNA FRASE
2: INTRODUCIR PALABRAS NUEVAS
3: COMUNICACION CON LA BASE DE CONOCIMIENTO
4: TERMINAR
OPCION: 1
DIGAME UNA FRASE
¿es Manuel un hombre?
(interrogativa)...
...
DIGAME UNA FRASE
¿Manuel es un hombre.
ERROR EN LA ESCRITURA DE LA FRASE
1. ES UNA FRASE INTERROGATIVA
```

```
2. ES UNA FRASE DECLARATIVA
3. ACABAR
2
(declarativa) ...
...
>batch;
PARA ACABAR LA INTRODUCCION DE FRASES ESCRIBIR: fin.
FRASE:
Manuel es un hombre.
FRASE:
Alicia es una mujer.
FRASE:
fin.
...
```

Test de reconocimiento de las palabras de la frase

En esta fase del proceso de analisis de frases se estudia si todas las palabras de la frase pertenecen al diccionario es decir si son "reconocidas" o no.

Se estudia si todas las palabras pertenecen al diccionario y si esto no ocurre se pregunta al usuario que tipo de error hay en cada una de las que no son reconocidas. Si el error es ortográfico el sistema pide al usuario la palabra correctamente escrita y la sustituye por la anterior. Si no pertenece al diccionario se pasa a la fase de introducción en el diccionario solicitando los datos pertinentes.

Hay todavia tres opciones más para recuperación de errores en esta fase, "sustituir" la palabra por un nuevo conjunto de palabras, "eliminar" la palabra sin que sea sustituida por otra y terminar en este punto el analisis de la frase. Como resultado del proceso se obtiene una secuencia de palabras que pertenecen al diccionario y ninguna información más.

```
>com-op;
SELECCIONE UNA OPCION :
    1: ANALISIS DE UNA FRASE
    2: INTRODUCIR PALABRAS NUEVAS
    3: COMUNICACION CON LA BASE DE CONOCIMIENTO
    4: TERMINAR
OPCION:  1
DIGAME UNA FRASE
Manual es el hombr.
ERROR EN LAS PALABRAS
Manual  hombr
SELECCIONE UNA DE LAS OPCIONES PARA CADA PALABRA:
    1. ERROR ORTOGRAFICO (SERA SUSTITUIDA)
    2. NO DEFINIDA
    3. SUSTITUIR (POR MAS DE UNA PALABRA)
    4. ELIMINAR
    5. ACABAR EL PROCESO DE LA FRASE
Manual  OPCION  1
hombr  OPCION  3
LA PALABRA Manual SE SUSTITUYE POR   Manuel
LA PALABRA hombr SE  SUSTITUYE  POR  (UNA  LISTA  ACABADA  EN
PUNTO)
hombre que viene.
LA FRASE PARA ANALIZAR ES:
Manuel  es  el  hombre  que  viene
```

Analisis sintáctico semántico

El método de analisis sintáctico-semántico ya se ha expuesto.

Detección y recuperación de errores de concordancia

Como ya se ha indicado el estudio de la concordancia entre las palabras que forman una frase es realizado a continuación de la construcción de las estructuras sintáctica

y semántica utilizando las reglas de la gramática y a pesar de que la definición de las restricciones sobre el genero y el número de las palabras vengan definidas en las reglas.
Una frase ademas de las dos estructuras anteriores tiene otra asociada y es la de un arbol con nodos etiquetados con listas, referidos a cada uno de los nodos "principales" de la estructura sintáctica, que indican los errores, si los hay, de genero y número en la parte correspondiente de la frase (se ha realizado un ejemplo detallado en el apartado).

Dentro de cada grupo de palabras que han de tener igual genero y/o número es la primera de ellas la que indica cual es el que deben tener todas las demas. No se ha realizado ningun modulo que eligiese segun unas reglas, una palabra del grupo para ser la indicadora del genero y el número ya que no se encontró ninguna razon convincete (ni estudio) sobre la mejora que produciria, por ejemplo, que en un grupo nominal formado por un articulo y un nombre comun fuera el nombre comun el que indica el genero y el número asociado al grupo nominal y no el articulo.

Si es detectado un error en la frase el sistema lo elimina de todas las formas posibles y es el usuario el que decide cual es la correcta. Se ve a continuación un ejemplo de una sesión sólo mostrando la parte correspondiente a la detección y recuperación de los errores de concordancia:

```
>com-op;
SELECCIONE UNA OPCION :
1: ANALISIS DE UNA FRASE
2: INTRODUCIR PALABRAS NUEVAS
3: COMUNICACION CON LA BASE DE CONOCIMIENTO
4: TERMINAR
OPCION:  1
DIGAME UNA FRASE
si Manuel viene y la nina es bonitas entonces Antonio sonrie.
ANALISIS SINTACTICO
```

...
ERROR DE CONCORDANCIA EN EL
numero
si Manuel viene yy la nina es bonita entonces Antonio sonrie
ES LA FRASE CORRECTA ?
si
...

(Ya se han visto ejemplos de esta parte en otros apartados).

Salida de estructuras asociadas a la frase

Las estructuras asociadas a una frase del lenguaje y reconocida como tal por el sistema contiene información que no se considera necesario exponer al usuario (fundamentalmente por perdida de claridad y dificultad de comprensión). Por ello existe un módulo que oculta esta información (entre ella el genero y numero de cada palabra, listas de errores de componentes,...) y que presenta en pantalla o papel el resultado de diversa forma (dibujo de arbol, lista Prolog, ...).

V.4.2 INTRODUCCION DE PALABRAS EN EL DICCIONARIO

En el diccionario cada palabra se representa mediante una clausula que contiene toda la información deseada, por lo tanto introducir nuevas palabras implica una modificación del diccionario al añadir una nueva clausula. El diccionario dispone de una parte general (con palabras pertenecientes a cualquier tipo de dominio) y de una parte "particular" que esta modularizada por grupos sintácticos.

Es posible añadir palabras al diccionario de forma interactiva siguiendo los pasos indicados por un conjunto de menus. Primero el sistema comprueba si la palabra ya está en el diccionario. Si es así aparece toda la información sobre ella y se puede elegir entre introducirla de nuevo con otras caracteristicas (rosa, nombre comun y adjetivo) o no hacerlo. En caso de continuar la introducción se pasa al módulo "añadir en el diccionario". Este módulo solicita sobre la palabra a introduir:

grupo sintáctico (np, nc, adj, ver)
genero (mas, fem) (si no es un verbo)
número (sin,plu) (si no es un nombre propio)
tipo (0, 1, 2) (si es un verbo)

La semántica de las palabras de las categorias (np, nc, ver, adj) a introducir está definida por el sistema y es independiente del tipo de semántica a utilizar. La introducción de otro tipo de palabras como conectivas o categorias que no sean (np, nc, ver, adj) se realiza siempre por el implementador. Segun la categoria sintáctica el módulo define la semántica de la palabra:

Si p es la palabra a introducir entonces la semántica es:

p si nombre comun
p si nombre própio

p si adjetivo

p si verbo proposicional

<p,x> si verbo(0)

<p,x,y> si verbo(1)

<p,x,y,z> si verbo(2)

Con la segunda parte de introducción de palabras al diccionario, módulo "introducir en diccionario" ademas se solicita al usuario las palabras con igual raiz pero con distinto genero y numero, salvo en el caso de introducción de un nombre própio que no se pregunta nada o de un verbo en el que no se admite variación segun el género.
Una vez recogida toda la información anterior se introduce la regla Prolog para esta palabra (y para las palabras correspondientes variando genero y/o número) así como la regla Prolog que las agrupa segun su categoría sintáctica.

Al terminar la introducción es el propio sistema el que almacena la nueva información en el fichero correspondiente.

Nota: No pueden ser introducidas palabras conteniendo la letra "ñ" o acentos por ser caracteres dobles (se introduce la palabra con la letra "n" en el primer caso). Tampoco pueden ser introducidas palabras con igual identificador (secuencia de letras que la forman) pero distintas caracteristicas en la misma sesión Prolog por problemas de la implementación de la versión utilizada.

Ejemplo de sesión de introducción de palabras en el diccionario:

```
>com-op;
SELECCIONE UNA OPCION :
1: ANALISIS DE UNA FRASE
2: INTRODUCIR PALABRAS NUEVAS
3: COMUNICACION CON LA BASE DE CONOCIMIENTO
4: TERMINAR
```

OPCION: 2

PALABRA A INTRODUCIR: Manuel

DEFINIR EL

GRUPO SINTACTICO (nc, adj, ver, np) np

GENERO (mas,fem,ambos): mas

SE INTRODUCEN EN EL DICCIONARIO.

...

PALABRA A INTRODUCIR: rosa

LA PALABRA rosa ESTA DEFINIDA COMO

nombre comun , singular , femenino

DESEA:

1: INTODUCIR rosa

2: NO INTRODUCIR rosa

OPCION: 1

DEFINIR EL

GRUPO SINTACTICO (nc, adj, ver, np) adj

GENERO (mas,fem,ambos): fem

NUMERO (sin,plu,ambos): sin

PALABRA CORRESPONDIENTE CON IGUAL RAIZ EN plural, femenino (NO, SI NO HAY)

rosas

PALABRA CORRESPONDIENTE CON IGUAL RAIZ EN singular, masculino (NO, SI NO HAY)

no

PALABRA CORRESPONDIENTE CON IGUAL RAIZ EN plural , masculino (NO, SI NO HAY)

no

¿ ES CORRECTO ? (SI, NO)

si

SE INTRODUCEN EN EL DICCIONARIO.

...

PALABRA A INTRODUCIR: pantano

DEFINIR EL

GRUPO SINTACTICO (nc, adj, ver, np) nc

GENERO (mas,fem,ambos): mes

MALA DEFINICION. SELECCIONE UNA OPCION:

1: DEFINIR pantano

2: TERMINAR LA DEFINICION

OPCION: 1

DEFINIR EL

GRUPO SINTACTICO (nc, adj, ver, np) nc

GENERO (mas,fem,ambos): mas

NUMERO (sin,plu,ambos): sin

PALABRA CORRESPONDIENTE CON IGUAL RAIZ EN singular , femenino (NO, SI NO HAY)

no

PALABRA CORRESPONDIENTE CON IGUAL RAIZ EN plural , femenino (NO, SI NO HAY)

no

PALABRA CORRESPONDIENTE CON IGUAL RAIZ EN plural , masculino (NO, SI NO HAY)

pantanos

¿ ES CORRECTO ? (SI, NO)

si

SE INTRODUCEN EN EL DICCIONARIO.

...

CAPITULO VI:
CONCLUSIONES Y ORIENTACIONES FUTURAS

VI CONCLUSIONES Y ORIENTACIONES FUTURAS

Se ha presentado en este trabajo un sistema de comprensión del lenguaje natural y de comunicación en castellano con sistemas basados en el conocimiento que aporta:

1. La integración de las diversas tareas de un sistema basado en el conocimiento con un módulo de tratamiento del castellano.

2. La adecuación a diferentes bases de conocimiento con una simple introducción interactiva de palabras relativas al dominio.

3. La definición de un subconjunto del castellano elegante y sencillo para la expresión del conocimiento. El algoritmo implementado que dirige el análisis del lenguaje es potente para la expresión de restricciones.

Las líneas de trabajo futuras se orientan a la ampliación o modificación del sistema Sirena para comunicación con sistemas basados en el conocimiento, de forma que:

1. Disponga de diferentes formalismos para la representación del conocimiento.

2. Realice la explicación del razonamiento seguido por el sistema para concluir una respuesta, con la misma metodología que la presentada.

3. Permita diálogos con el usuario dado que la actuación de un sistema basado en el conocimiento se entiende como un continuo compartir información.

BIBLIOGRAFIA

[Abramson, 84]
"Definite clause translation grammars and the local specification of data types as unambiguous free grammars"
Proceedings of the international conference on fifh generation computer systems, 1984.

[Bates, 78]
Bates M.
"The theory and Practice of Augmented Transition network grammars"
M Bates. Natural language communication with computers nro 63. Ed L. Bolc, Springer-Verlag 1978.

[Bernth, 85]
"Logic Programed Natural Language Understanding"
Master Thesis. Institute of Datalogy. University of Copenhagen.

[Bernth, 85]
"Natural Language Translation by means of Prolog"
Proc. SAIS 1986

[Boyer, Lapalme, 84]
Boyer M. Lapalme G.
"Generating Sentences from Semantic Networks"
Rennes 1984

[Bruce, 75]
Bruce, B. C.
Case systems for natural language
AI 6, pp327-360, 1975

[Carbonell 78]
Carbonell J.R.
"Politics, automated ideological reasoning"
Cognitive Science 2 1978.

[Carbonell,Hayes 83]
Carbonell J.G. ,Hayes P.
"Natural Lanquaqe Tutorial"
IJCAI-83

[Clocksin,Mellish, 81]
Clocksin W.F. Mellish C.S.
"Programminq in Prolog" Springer-Verlaq, New York 1981.

[Colby et all, 77]
Colby, K., Parkison,R.,Faught,W.
Conversational lanquaqe comprehension usinq integrated Pattern-Matchinq and Parsinq.
Artificial intelligence 9 (1977), pp 11-134.

[Colmerauer, 70]
Colmerauer A.
"Les systemes-Q ou un formalisme pour analizer et synthetiser des phrases sur ordinateur"
Publication interne nro 43. Departament d'informatique. Faculté des sciences. Université de Montreal. Sep 1970.

[Colmerauer, 75]
"Metamorphosis qrammars"
GIA de Marsella o en
Natural lanquaqe communication with computers nro 63.
Ed L. Bolc, Springer-Verlaq 1978.

[Colmerauer, 77]
"Un sous-ensemble interessant du francais"
GIA. Marsella 1977.

[Colmerauer, 83]
"Prolog II in ten figures"
Proc. IJCAI 83, Karlsruhe 1983.

[Colmerauer et Kittredge, 83]
"ORBIS"
GIA Marsella 1983.

[Coulon, 85]
D. Coulon
"Examen de diverses methodes utilisées en representation des connaissances"
Curso Universidad del Pais Vasco, San Sebastian 1985.

[Chomsky, 57]
Chomsky, N.
"Estructuras sintacticas"
(Ed en castellano siglo veintiuno, 1974)
Mouton. The Haque, 1957.

[Chomsky, 65]
"Aspects of the theory of syntax"
Cambridge MIT Press 1965.

[Cuena, 85]
Cuena J.
"Lógica Informática" Alianza-Informática, Madrid 1985.

[Cuena,García-Serrano, 87]
"An expert system for finantial risk evaluation and decision making". Proc. The 7th International Workshop: Expert Systems and their applications, Avignon 87, Francia 1987.

[Dahl 77]
"Un systeme deductif d'interrogation de banques de données en espagnol"
Tesis de Tercer Ciclo. Gia. Marsella 1977.

[Dahl, McCord 83]
"Treating coordination in logic grammars"
AJCL, vol 9, Num 2 June 1983.

[Dahl, Abramson 84]
Dahl V., Abramson H.
"On gapping grammars" Second international conference on logic grammars. Uppsala Sweden 1984.

[Dahl, 84]
"More on Gapping Grammars "
Proc. International Conference on Fifth Generation Computer Systems, 1984.

[Feigenbaum, 81]
A. Barr, E. Feigenbaum
The handbook of Artificial Intelligence.
Pitman Books Ed. 1981.

[Fillmore, 68]
"The case for case"
In Bach, E. and Harms, R.T. (eds)
Universals in linguistic theory. New York, 1968.

[García-Serrano 85]
García-Serrano A.
"Programación lógica en la comprensión del lenguaje Natural"
Proc. I Jornadas de Inteligencia Artificial en la Universidad Complutense de Madrid. 1985 .

[Giannesini et all, 85]
Giannesini M. Kanoui H. Pasero R. Van Caneghem M.
"Prolog" InterEditions 1985.

[Hayes-Carbonell, 83]
P. Hayes, J. Carbonell
Natural language tutorial.
IJCAI 1983 .

[Hendrix 77]
"Human engineering for applied natural language processing"
Proc. Fifth IJCAI 1977.

[Hendrix, 83]
Natural Language interface (Tutorial) AAAI-83.

[Hendrix-Sacerdoti, 81]
G. Hendrix, E. Sacerdoti
"Natural Language procesing.The field in perspective"
SRI Internacional, tech. Note 237, 1981.

[Hernandez 83]
Hernandez Irurzun L.
"Interprete de gramáticas de redes de transición aumentadas".
(Trabajo para obtención de grado) San Sebastian 1983.

[Hopcroft, Ullman 69]
"Formal Languages and their applications to automata"
Addison Wesley. 1969.

[Kowalsky, 79]
"Logic for Problem solving"
Elsevier North-Holland 1979.

[Oliveira, Pereira et Sabatier 82]
"URBI an expert system for environmental ressources evaluation through natural language"
First International Logic Programming Conference. Marsella 1982.

[O'Shea, M.Eisenstadt 84]
"Artificial Intelligence, Tools Techniques and Applications"
Harper and Row Publishers, New York, 1984.

[Pasero, 82]
Pasero R.
"Dialoque in natural lanquaqe"
First International Logic Programminq Conference. Marsella 1982.

[Pentonnen, M. 85]
Pentonnen, M.
"A small program understandinq finish"
University of Turku. Finland 1985.

[Pereira, 81]
Pereira, F.
"Extraposition Grammars"
American Journal of Computational Linguistics, V 7, nro 4, October-December 1981.

[Pereira, Warren 80]
"Definite Clause Grammars for Language Analysis"
AI nº13, 1980.

[Pereira, 83]
SRI. Technical Note 275. 1983.

[Pereira-Lopes 84]
"Implementinq Dialoques in a knowledqe information system"
Proc. Natural languaje understandinq and logic programminq. Rennes 1984.

[Pique,Sabatier, 82]
"An informative, adaptable and efficient natural language consultable database system"
Proc. of ECAI 1982.

[Popov, 86]
Talkinq with Computers in Natural Lanquaqe.
Springer-Verlaq 1986.

[Ritchie, 84]
G. Ritchie
Tutorial on Natural Language procesing
ECAI 84.

[Roussel, 75]
Roussel P.
"Prolog, manuel de Référence et d'utilisation"
GIA Marseille 1975.

[Sabatier, Sedogbo, 84]
Sabatier P., Sedogbo C.
"Les gramaires logiques"
Actes du colloque Traitement Automatique du Langage Naturel
Universite de Nantes. Junio 1984.

[Sabatier 84]
"Puzzle grammars"
Proc. Natural language understanding and logic programming.
Rennes 1984.

[Sacerdoti, 77]
"Language access to distributed data with error recovery"
Fifth IJCAI 1977.

[Saint-Dizier 84]
Saint Dizier P.
"Quantifier hierarchy in a semantic representation of natural language sentences"
Proc. Natural language understanding and logic programming.
Rennes 84.

[Sedogbo, 84]
Sedogbo, C.
"A Meta grammar for handling coordination in logic grammars"
Proc. Natural language understanding and logic programming.
Rennes 1984.

[Sopeña, 83]
Sopeña, L.
"Aportaciones a la comprensión del castellano en un sistema de interrogación de bases de datos relacionales" Tesis Doctoral, Universidad del Pais Vasco. 1983.

[Sowa, 84]
"Conceptual structures: Information processing in Mind and Machine"
Addison-Wesley. 1984

[Schank, 72]
"Conceptual dependency: A theory of Natural Language"
Cognitive Psichology 3(4), pp 552-630, 1972

[Schank,Abelson, 77]
"Scripts, plans, goals and understanding. Hillsdale, N.J.: Lawrence Erlbaum.

[Verdejo, 80]
Verdejo F.
"Un analizador semántico para un sistema pregunta-respuesta en lenguaje Natural". Tesis Doctural, Universidad Complutense de Madrid, 1980

[Waltz, 85]
Waltz D.
"Natural Language Processing"
Tutorial, IJCAI 85.

[Warren, Pereira, 82]
Warren D., Pereira F.
"An efficient easily adaptable system for interpreting natural language queries"
Americal journal of computational linguistics 8,3-4,1982

[Weizenbaum, 66]
Weizenbaum, J.
"ELIZA, A computer Program for the study of natural language communication between man and machine". Comm ACM 9,1 (Jan,1966), pp 36-45.

[Wilks, 75]
Wilks, Y.
"An intelligent analyzer and understander of english"
CACM 18 pp 264-274, 1975.

[Wilks, 75]
Wilks, Y.
"Preference Semantics"
In Formal semantics of natural language analysis, Keenan, Ed.
Cambridge University Press, 1975.

[Wilks, 84]
Wilks, Y.
"Preference Semantics"
IBM Europe Institute, Davos, Suiza 1984.

[Winograd, 72]
Winograd, T.
"Understanding Natural Language"
Academic Press 1972.

[Winograd, 83]
"Natural Language as a cognitive process. Addison Wesley 1983.

[Woods, 70]
Woods, W.
"Transition network grammars for natural language analysis"
Comm ACM 13, 10 (oct 1970), pp 591-606.

APENDICE 1:
SESION EJEMPLO

APENDICE 1: SESION EJEMPLO

Se realiza a continuación un ejemplo completo y breve para mostrar al sistema Sirena en comunicación con una base de conocimiento. No se explica, como ya se ha dicho en otro capítulo, el tipo de proceso que maneja el conocimiento para su representación en la base o el tipo de motor de inferencia utilizado para responder a las preguntas, por no ser el objetivo de esta tesis.

Se construye la base de conocimiento a partir de las siguientes informaciones:

Información general:

I1 : Los bancos conceden un crédito si el límite del crédito es bajo.

I2 : Si el banco concede el crédito entonces el crédito es concedido.

Información sobre los bancos:

I3 : Banesto es un banco.

I4 : Bansander es un banco.

I5 : Si la coyuntura es estable y el límite del crédito es alto entonces Banesto concede el crédito.

I6 : Bansander no concede un crédito si el límite del crédito es alto.

Información sobre los créditos:

I7 : Credun es un crédito.

I8 : Credos es un crédito.

I9 : El límite de Credun es alto.

I10: Credos tiene límite bajo.

Información general del momento en estudio:

I11: La coyuntura es estable.

La sesión será:

```
>com-op;

SELECCIONE UNA OPCION :

     1: ANALISIS DE UNA FRASE
     2: INTRODUCIR PALABRAS EN EL DICCIONARIO
     3: COMUNICACION CON LA BASE DE CONOCIMIENTO
     4: TERMINAR

  OPCION:  3

SELECCIONE UNA OPCION:

  1: INTRODUCIR CONOCIMIENTO
  2: PREGUNTAR A LA BASE

OPCION: 1

 DIGAME

Credun es un credito.

ERROR EN LAS PALABRAS

Credun

SELECCIONE UNA DE LAS OPCIONES PARA CADA PALABRA:

1. ERROR ORTOGRAFICO (SERA SUSTITUIDA)
2. NO DEFINIDA
3. SUSTITUIR (POR MAS DE UNA PALABRA)
4. ELIMINAR
5. ACABAR EL PROCESO DE LA FRASE

Credun  OPCION  2

PALABRA PARA INTRODUCIR:   Credun

DEFINIR EL
             GRUPO SINTACTICO (nc, adj, ver, np) np
             GENERO (mas,fem,ambos):          mas

SE INTRODUCEN EN EL DICCIONARIO.

ANALISIS SINTACTICO
   ...

 ANALISIS SEMANTICO

credito
   |
 Credun
```

SELECCIONE UNA OPCION :

...

DIGAME

Credos es un credito.

ERROR EN LAS PALABRAS

Credos

SELECCIONE UNA DE LAS OPCIONES PARA CADA PALABRA:

...

ANALISIS SEMANTICO

credito

|

Credos

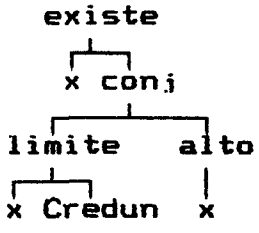
SELECCIONE UNA OPCION :

...

DIGAME

el limite de Credun es alto.

ANALISIS SEMANTICO



SELECCIONE UNA OPCION :

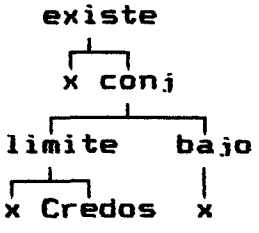
...

DIGAME

Credos tiene limite bajo.

...

ANALISIS SEMANTICO



SELECCIONE UNA OPCION :

...

DIGAME

Bansander es un banco.

ERROR EN LAS PALABRAS

Bansander

SELECCIONE UNA DE LAS OPCIONES PARA CADA PALABRA:

...

ANALISIS SEMANTICO

banco

|

Bansander

SELECCIONE UNA OPCION :

...

DIGAME

Banesto es un banco.

ERROR EN LAS PALABRAS

Banesto

SELECCIONE UNA DE LAS OPCIONES PARA CADA PALABRA:

...

ANALISIS SEMANTICO

banco

|

Banesto

SELECCIUNE UNA OPCION :

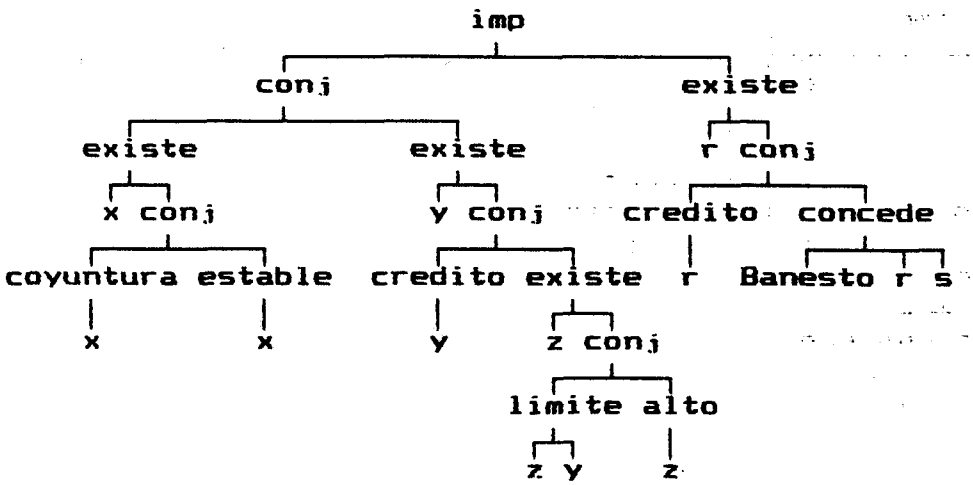
...

DIGAME

**si la coyuntura es estable y el limite del credito es alto,
Banesto concede el credito.**

...

ANALISIS SEMANTICO



SELECCIUNE UNA OPCION :

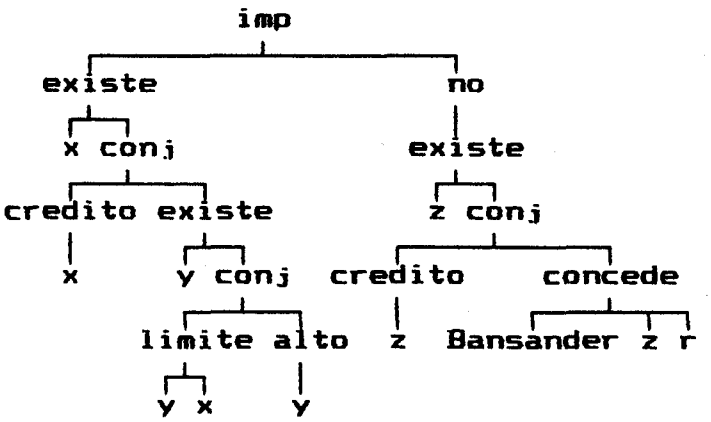
...

DIGAME

**Bansander no concede un credito si
el limite del credito es alto.**

...

ANALISIS SEMANTICO



SELECCIONE UNA OPCION :

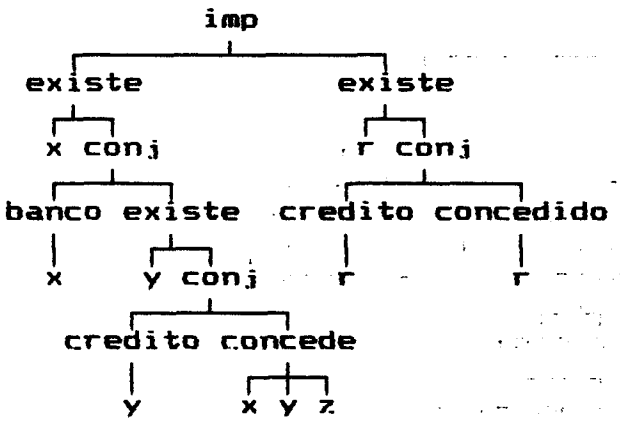
...

DIGAME

si el banco concede el credito entonces el credito es concedido.

...

ANALISIS SEMANTICO



SELECCIONE UNA OPCION :

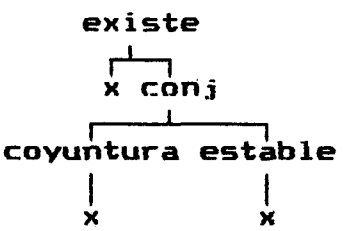
...

DIGAME

la coyuntura es estable.

...

ANALISIS SEMANTICO

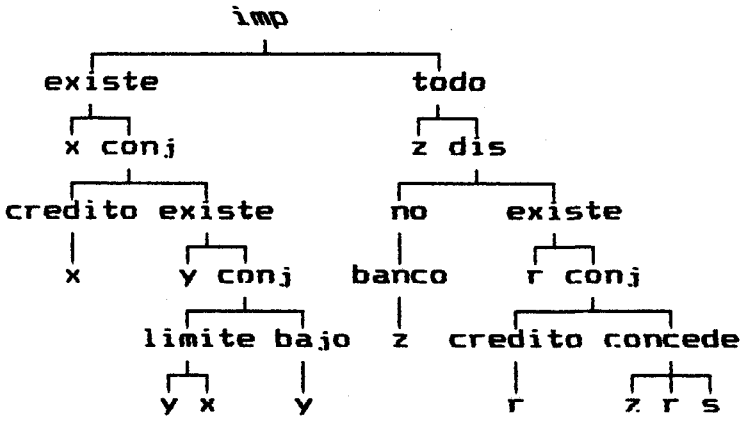


ECCIONE UNA OPCION :

AME

bancos conceden un credito
l limite del credito es bajo.

ANALISIS SEMANTICO



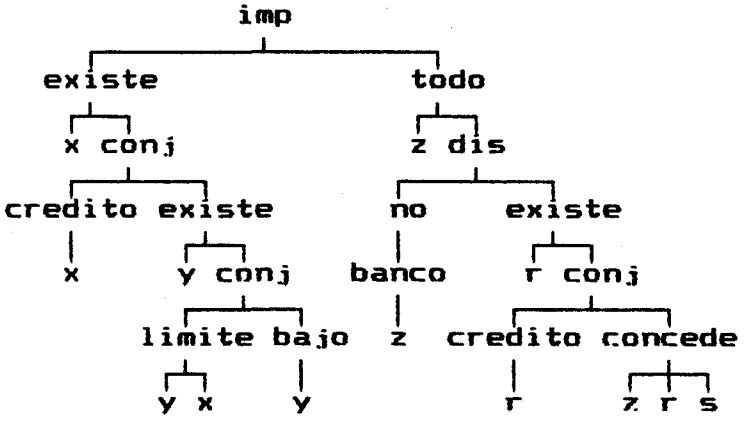
SELECCIONE UNA OPCION :

DIGAME

los bancos conceden un credito
si el limite del credito es bajo.

...

ANALISIS SEMANTICO



El conocimiento anterior recogido por Sirena se almacena en la base de conocimiento con las siguientes clausulas Prolog:

```
no(concede(Bansander,y,z)) -> conj(limite(x,y),alto(x))
                              credito(y);
concede(Banesto,y,s) -> conj(coyuntura(x),estable(x))
                        credito(y) conj(limite(z,y),alto(z));
concede(x,z,s) -> banco(x) credito(z)
                  conj(limite(r,z),bajo(r));

concedido(x) -> banco(y) credito(x) concede(y,x,s);

banco(Banesto) ->;
banco(Bansander) ->;

credito(Credun) ->;
credito(Credos) ->;

conj(coyuntura(x),estable(x)) ->;
conj(limite(x,Credos),bajo(x)) ->;
conj(limite(x,Credun),alto(x)) ->;
```

Este conocimiento será utilizado por un pequeño motor de inferencia para responder a prequntas de los tipos:
Prequnta "si-no" que tiene por posibles respuestas a "si", "no", "no se nada mas",
Prequntas que esperan por respuesta un conjunto de objetos que verifican la condición expresada en ellas. Las respuestas del sistema son la lista de los objetos solicitados, "nadie que yo sepa" o "no se nada mas".

La sesión de prequnta a la base podrá ser como la que se muestra a continuación:

```
>com-op:

SELECCIONE UNA OPCION :

      1: ANALISIS DE UNA FRASE
      2: INTRODUCIR PALABRAS EN EL DICCIONARIO
      3: COMUNICACION CON LA BASE DE CONOCIMIENTO
      4: TERMINAR

  OPCION:  3

SELECCIONE UNA OPCION:

  1: INTRODUCIR CONOCIMIENTO
  2: PREGUNTAR A LA BASE

OPCION: 2

 DIGAME

¿quien concede Credos?

ANALISIS SEMANTICO:

quien
┌──┐
x concede
  ┌──┬──┐
  x Credos y

RESPUESTA: Banesto Bansander

SELECCIONE UNA OPCION :
  ...

 DIGAME

¿que banco concede Credun?

ANALISIS SEMANTICO:

  que
  ┌──┐
  x conj
  ┌──┬──┐
banco concede
  │   ┌──┬──┐
  x   x Credun y

RESPUESTA: Banesto
```

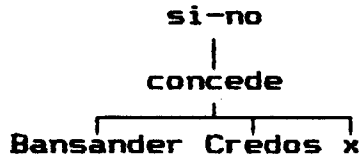
SELECCIONE UNA OPCION :

...

DIGAME

¿concede Bansander Credos?

ANALISIS SEMANTICO:



RESPUESTA: **no**

SELECCIONE UNA OPCION :

...

DIGAME

es un credito Bansander?

ANALISIS SEMANTICO

RESPUESTA: **no se nada mas**

SELECCIONE UNA OPCION :

...

DIGAME

es un banco Banesto?

ANALISIS SEMANTICO:

RESPUESTA: **si**

APENDICE 2:
PROGRAMAS DEL SISTEMA SIRENA

```
!
PREPARA MEMORIA PROLOG PARA SISTEMA

(Mundo Base)
!

preparar ->
        kill-subworld("Interface")
        kill-subworld("Normal")
        new-subworld("Normal",3500)
        insert("auxi.pro")
        outml("FICHERO AUXILIAR")
        outml("CONTINUE CON inc1;");

inc1 -> new-subworld("Dic",9000)
        insert("dic.pro")
        outml("FICHERO CON EL DICCIONARIO")
        inc2;

inc2 -> new-subworld("Gram",12000) insert("gram.pro")
        outml("FICHERO CON LA GRAMATICA")
        inc3;

inc3 -> new-subworld("Gen",12000) insert("gen.pro")
        outml("FICHERO GENERAL")
        outml("CONTINUE CON com-op;");

subir -> world("Normal") outml("NO SE PUEDE");
subir -> world("Dic") climb world(x) outml(x);
subir -> world("Gram") climb world(x) outml(x);
subir -> world("Gen") climb world(x) outml(x);

bajar -> world("Gen") outml("NO SE PUEDE");
bajar -> world("Gram") down("Gen") world(x) outml(x);
bajar -> world("Dic") down("Gram") world(x) outml(x);
bajar -> world("Normal") down("Dic") world(x) outml(x);

        ;
```

```
!
 SELECCION DE TAREA
!

com-op ->
    line
    outml("SELECCIONE UNA OPCION :")
    line
    blancos(4) outml(" 1: ANALISIS DE UNA FRASE") blancos(4)
    outml(" 2: INTRODUCIR PALABRAS EN EL DICCIONARIO")
    blancos(4)
    outml(" 3: COMUNICACION CON LA BASE DE CONOCIMIENTO")
    blancos(4) outml(" 4: TERMINAR")
    line outm("  OPCION:  ")
    in-integer(x)
    line
    dec-com(x);

dec-com(1) -> / anal;
dec-com(2) -> / in-pal;
dec-com(3) -> / com-base;
dec-com(4) ->
    line outml("  ADIOS") fallo;

anal ->
    line
    outml(" DIGAME UNA FRASE ")
    line int-frase(x)
    com(x,y,o) / com-op;
anal ->
    line outml("FRACASO EN EL ANALISIS DE LA FRASE") line
    com-op;

in-pal ->
    line outm("PALABRA A INTRODUCIR:  ")
    in-ident(x)
    anadir-dicc(x) com-op;

com-base ->
    line
    dec-int-preg-base(d)
    outml(" DIGAME")
    line int-frase(x)
    com(x,y,z)/
    resto-int-o-preg(d,z);
com-base ->
    line outml("FRACASO EN EL ACCESO A LA BASE") line
    com-op;

dec-int-preg-base(d) ->
    outml("SELECCIONE UNA OPCION:") line
    blancos(2) outml("1: INTRODUCIR CONOCIMIENTO")
    blancos(2) outml("2: PREGUNTAR A LA BASE") line
    outm("OPCION: ") in-integer(d) line;

resto-int-o-preg(d,z) ->
    int-o-resp(d,z) / com-op;
```

```
resto-int-o-preg(d,z) ->
   line
   outml("FRACASO EN EL ACCESO A LA BASE DE CONOCIMIENTO")
   line com-op;

int-o-resp(1,s) ->;
   !modulo-representacion-conocimiento(s,s')
   introducir-en-base(s);!
int-o-resp(2,s) ->
   preguntar-base(s,r);

!
```

```
 PROCESO DE ANALISIS DE LA FRASE
!

com(y,i,o) ->
   det-tipo-frase(y,l',t)
   test-recon-pal(l',l)
   llamada-analisis(l,i,o,t) ;

det-tipo-frase(y,l,t) ->
   elemento("¿",y) /
   quitar("¿",y,l')
   cont-int(l',l,t);
det-tipo-frase(y,l,int) ->
   elemento("?",y) /
   quitar("?",y,l);
det-tipo-frase(y,l,dec) ->
   elemento(".",y) /
   quitar(".",y,l);

cont-int(l',l,int) ->
   elemento("?",l') /
   quitar("?",l',l);
cont-int(l',l,t) ->
   elemento(".",l') /
   quitar(".",l',l)
   preg-tipo-frase(t);

preg-tipo-frase(t) ->
   line outml("  ERROR EN LA ESCRITURA DE LA FRASE ")
   line outml("  1. ES UNA FRASE INTERROGATIVA ")
   outml("  2. ES UNA FRASE DECLARATIVA ")
   outml("  3. ACABAR EL PROCESO DE ANALISIS") line blancos(2)
   in-integer(r) resp-tipo-frase(r,t);

resp-tipo-frase(1,int) -> /;
resp-tipo-frase(2,dec) ->;
resp-tipo-frase(3,t) -> / fallo;
resp-tipo-frase(x,t) ->
   preg-tipo-frase(t) ;

llamada-analisis(l,i,o,t) ->
   llana(l,i',o',t,s)
   cont-llana(l,i,o,t,s,i',o');

cont-llana(l,i,o,t,s,i',o') ->
   no-iguales(l,s) /
   llamada-analisis(s,i,o,t);
cont-llana(l,i,o,t,l,i,o) ->
   salida-analisis(t,l,i,o);

llana(l,i,o,dec,s) ->
   anal-dec(l,i,o,<t,e>)
   detec-err-con(e)
   concordancia(e,l,s);

llana(l,i,o,int,s) ->
   anal-int(l,i,o,<t,e>)
```

```
   detec-err-con(e)
   concordancia(e,l,s);

anal-int(l,i,o,e) ->
   frontera(<e,i>,l)
   frase-int(<e,i>,o);

anal-dec(l,i,o,e) ->
   frontera(<e,i>,l)
   frase-declarativa(<e,i>,o);

salida-analisis(t,l,i,o) ->
   line
   outm("ANALISIS SINTACTICO")
   sta(t) line
   dibuja-arbol-sint(i,i') line
   outml("ANALISIS SEMANTICO")
   line outl(o)
   bottom assert(frase(t,l,i',o),nil);

sta(dec) -> line;
sta(int) -> outml(" INTERROGATIVO");

!
 LECTURA DE LA FRASE
!

int-frase(x) ->
   in-sentence(y,s)
   filtro(y,x);

filtro(nil,nil) -> /;
filtro(".".x,".".y) -> / filtro(x,y);
filtro("¿".x,"¿".y) -> / filtro(x,y);
filtro("?".x,"?".y) -> / filtro(x,y);
filtro("a".x,aa.y) -> / filtro(x,y);
filtro("y".x,yy.y) -> / filtro(x,y);
filtro("o".x,oo.y) -> / filtro(x,y);
filtro(",".x,cc.y) -> / filtro(x,y);
filtro(a.x,b.y) ->
   string-integer(a,b) /
   filtro(x,y);
filtro(a.x,b.y) ->
   string-ident(a,b) /
   filtro(x,y);
filtro(c.x,c.y) ->
   filtro(x,y);

!
```

```
   detec-err-con(e)
   concordancia(e,l,s);

anal-int(l,i,o,e) ->
   frontera(<e,i>,l)
   frase-int(<e,i>,o);

anal-dec(l,i,o,e) ->
   frontera(<e,i>,l)
   frase-declarativa(<e,i>,o);

salida-analisis(t,l,i,o) ->
   line
   outm("ANALISIS SINTACTICO")
   sta(t) line
   dibuja-arbol-sint(i,i') line
   outml("ANALISIS SEMANTICO")
   line outl(o)
   bottom assert(frase(t,l,i',o),nil);

sta(dec) -> line;
sta(int) -> outml(" INTERROGATIVO");

!
 LECTURA DE LA FRASE
!

int-frase(x) ->
   in-sentence(y,s)
   filtro(y,x);

filtro(nil,nil) -> /;
filtro(".".x,".".y) -> / filtro(x,y);
filtro("¿".x,"¿".y) -> / filtro(x,y);
filtro("?".x,"?".y) -> / filtro(x,y);
filtro("a".x,aa.y) -> / filtro(x,y);
filtro("y".x,yy.y) -> / filtro(x,y);
filtro("o".x,oo.y) -> / filtro(x,y);
filtro(",".x,cc.y) -> / filtro(x,y);
filtro(a.x,b.y) ->
   string-integer(a,b) /
   filtro(x,y);
filtro(a.x,b.y) ->
   string-ident(a,b) /
   filtro(x,y);
filtro(c.x,c.y) ->
   filtro(x,y);

!
```

```
 INTRODUCCION DE FRASES EN BATCH
¦

batch ->
   paper line
   outml("PARA ACABAR ESCRIBIR: fin.")
   line int-batch(1)
   anal-batch(1)
   no-paper
   quit;

int-batch(<x>.1) ->
   line outml("FRASE: ") line
   int-frase(x)
   fin-batch(x,1) /;
int-batch(nil) ->;

fin-batch(fin.".".nil,1) ->
   / fallo;
fin-batch(x,1) ->
   int-batch(1);

anal-batch(<x>.nil) ->
   /
   line outml("..page") line
   line outml(" FRASE: ") line
   sal-lista1(x) line
   com-batch(x);
anal-batch(<x>.r) ->
   line outml("..page") line
   line outml(" FRASE: ") line
   sal-lista1(x) line
   com-batch(x)
   anal-batch(r);

com-batch(y) ->
   com(y,i,o);

¦
 GUARDAR EL ANALISIS REALIZADO PARA PROCESO CON DESSIN
¦

guardar ->
   output("frases.pro")
   top find-rule(frase(t,l,i,o))
   list(999)
   output("console") close-output
   outml("FRASES PARA PROCESO CON DESSIN") line;

¦
```

INTRODUCCION DE PALABRAS EN EL DICCIONARIO

```
(anadir-dicc siempre es verdad)
!

anadir-dicc(x) ->
   esta-dicc(x) /;
anadir-dicc(x) ->
   an-dicc(x,c,n.g,no-en-dicc) /;
anadir-dicc(x) ->
   line
   outm("NO SE INTRODUCE.")
   outl(x);

esta-dicc(x) ->
   lpi(x,l)
   dif(l,nil) /
   menu2-an-dicc(x,l,en-dicc);
esta-dicc(x) ->
   fallo;

an-dicc(x,c,n.g,no-en-dicc) ->
   menu-an-dicc(x,c,n.g) /
   int-dicc(x,c,n.g);
an-dicc(x,c,n.g,en-dicc) ->
   menu-an-dicc(x,c,n.g) /
   int-dicc-ya(x,c,n.g);
an-dicc(x,c,n.g,e) ->
   menu3-an-dicc(x,e);

menu-an-dicc(x,c,n.g) ->
   line outml("DEFINIR EL")
   blancos(12) outm("GRUPO SINTACTICO ")
   outm("(nc, adj, ver, np) ") in-ident(c') gr-sint(c') line
   segun(c',n.g,c);

segun(np,sin.g,np) ->
   / blancos(12) outm("GENERO (mas,fem,ambos):")
   blancos(8) in-ident(g') gen(g')
   camb-ambos(g',g) line;
segun(ver,n.g,<ver,x>) ->
   / blancos(12) outm("NUMERO (sin,plu,ambos):")
   blancos(8) in-ident(n') num(n')
   camb-ambos(n',n) line
   blancos(12) outm("TIPO (0,1:(CD),2:(CD,CI),3:PROP):")
   blancos(8) in-integer(y) tipo-ver(y,x) line;
segun(c,n.g,c) ->
   blancos(12) outm("GENERO (mas,fem,ambos):")
   blancos(8) in-ident(g') gen(g')
   camb-ambos(g',g) line
   blancos(12) outm("NUMERO (sin,plu,ambos):")
   blancos(8) in-ident(n') num(n')
   camb-ambos(n',n) line;
```

```
menu3-an-dicc(x,e) ->
   line
   outm("MALA DEFINICION. ")
   outm1("SELECCIONE UNA OPCION: ")
   line outm("  1: DEFINIR ") out1(x)
   outm1("  2: TERMINAR LA DEFINICION")
   line
   outm("OPCION: ")
   in-integer(s) dec3-an(s,x,e);

dec3-an(1,x,e) -> /an-dicc(x,c,n.g,e);
dec3-an(2,x,e) -> /fallo;
dec3-an(x,y,e) -> menu3-an-dicc(y,e);

menu2-an-dicc(x,l,e) ->
   line outm("LA PALABRA ") out(x)
   outm1(" ESTA DEFINIDA COMO ")
   lpi-sal(l)
   outm1("DESEA: ") line
   outm("  1: INTODUCIR ") out1(x)
   outm("  2: NO INTRODUCIR ") out1(x)
   line outm("OPCION: ")
   in-integer(r)
   dec2-an-dicc(r,x,l,e);

dec2-an-dicc(1,x,l,e) ->
   / an-dicc(x,c,n.g,e);
dec2-an-dicc(2,x,l,e) ->
   / line outm1("NO SE INTRODUCE EN EL DICCIONARIO");
dec2-an-dicc(u,x,l,e) ->
   menu2-an-dicc(x,l,e);

lpi(x,<nc,n.g>.l) ->
   <x,nc,n.g,s> /
   lpii(x,l);
lpi(x,l) ->
   lpi1(x,l);
lpi1(x,<adj,n.g>.l) ->
   <x,adj,n.g,s> /
   lpi2(x,l);
lpi1(x,l) ->
   lpi2(x,l);
lpi2(x,<np,n.g>.nil) ->
   <x,np,n.g,s> /;
lpi2(x,nil) ->;

camb-ambos(ambos,n) -> /;
camb-ambos(n,n) ->;

gr-sint(c) -> grupos-sintacticos(l) pertenece'(c,l) /;
gr-sint(c) -> fallo;

tipo-ver(x,x) -> pertenece'(x,0.1.2.nil) /;
tipo-ver(x,prop) -> bound(x) /;
tipo-ver(x,y) -> fallo;
```

```
gen(g) -> pertenece'(g,fem.mas.ambos.nil) /;
gen(g) -> fallo;

num(n) -> pertenece'(n,sin.plu.ambos.nil) /;
num(n) -> fallo;

pertenece'(x,l) ->
   bound(x) /
   pertenece(x,l);
pertenece'(x,l) ->;


lpi-sal(<c,n.g>.l) ->
   line trad-gs(c) outm(", ")
   trad-ng(n,g)
   lpi-sal(l);
lpi-sal(nil) ->
   line line;

¡no se pueden anadir palabras (debido a manejo Prolog
de identificadores) que hayan sido definidas con otras
caracteristicas en esa misma sesion¡

int-dicc-ya(x,c,n.g) ->
   sem(x,c,s)
   pal-ng(<n.g,x>,l,c)
   climb climb
   anadir-todas(l,c,s)
   assert(<c,l>,nil)
   output("dic.pro") top list(9999)
   output("console") close-output
   down("Gram") down("Gen")
   line outml("SE INTRODUCEN EN EL DICCIONARIO.") line;

int-dicc(x,c,n.g) ->
   sem(x,c,s)
   pal-ng(<n.g,x>,l,c)
   climb climb
   anadir-todas(l,c,s)
   assert(<c,l>,nil)
   output("dic.pro") top list(9999)
   output("console") close-output
   down("Gram") down("Gen")
   anadir-todas(l,c,s)
   line outml("SE INTRODUCEN EN EL DICCIONARIO.") line;

anadir-todas(nil,t,s) ->;
anadir-todas(<n.g,x>.l,t,s) ->
   assert(<x,t,n.g,s>,nil)
   anadir-todas(l,t,s);

pal-ng(<n.g,x>,<n.g,x>.nil,np) -> /;
pal-ng(<n.g,x>,l,c) ->
   pal-ng'(<n.g,x>,l',c)
   blancos(2) outml("¿ ES CORRECTO ? (SI, NO)")
   blancos(4) in-ident(i) line
   dec-pal-ng(i,<n.g,x>,l,c,l');
```

```
pal-ng'(<n.g,x>,l,<ver,t>) ->
   / outm("VERBO EN ") dif-n(n,n') trad-n(n')
   blancos(2) in-ident(y) line
   quit-est(<p,no>,<n.g,x>.<n'.g,y>.nil,l);
pal-ng'(<n.g,x>,<n.g,x>.nil,c) ->
   free(n) free(g) /;
pal-ng'(<n.g,x>,m,c) ->
   free(n) /
   outm("PALABRA CON IGUAL RAIZ EN ") dif-g(g,g') trad-g(g')
   outml(" (NO, SI NO EXISTE)")
   blancos(2) in-ident(y) line
   quit-est(<p,no>,<n.g,x>.<n.g',y>.nil,m);
pal-ng'(<n.g,x>,m,c) ->
   free(g) /
   outm("PALABRA CON IGUAL RAIZ EN ") dif-n(n,n') trad-n(n')
   outml(" (NO, SI NO EXISTE)")
   blancos(2) in-ident(y) line
   quit-est(<p,no>,<n.g,x>.<n'.g,y>.nil,m);
pal-ng'(<n.g,x>,<n.g,x>.l,c) ->
   lis-ng(s)
   quitar((n.g),s,s')
   preg-ng(x,s',l')
   quit-est(<p,no>,l',l);

dec-pal-ng(si,<n.g,x>,l,c,l) -> /;
dec-pal-ng(no,<n.g,x>,l,c,l') ->
   pal-ng(<n.g,x>,l,c);

dif-n(sin,plu) ->/;
dif-n(plu,sin) ->;
dif-g(fem,mas) ->/;
dif-g(mas,fem) ->;

preg-ng(x,nil,nil) -> /;
preg-ng(x,(n.g).l,<n.g,y>.l') ->
   outm("PALABRA CON IGUAL RAIZ EN ") trad-ng(n,g)
   outml(" (NO, SI NO HAY)")
   blancos(2) in-ident(y) line
   preg-ng(x,l,l');

quit-est(<x,a>,l,l') ->
   quitar(<x,a>,l,l'') /
   quit-est(<y,a>,l'',l');
quit-est(<x,a>,l,l) ->;

trad-gs(nc) ->
   / outm("nombre comun ");
trad-gs(np) ->
   / outm("nombre propio");
trad-gs(adj) ->
   / outm("adjetivo ");
trad-gs(ver) ->
   / outm("verbo");
trad-gs(x) ->
   outml("error traduciendo grupo sintactico");
```

```
trad-n(x) ->
   free(x) /
   outm("singular y plural");
trad-n(sin) ->
   / outm("singular");
trad-n(plu) ->
   / outm("plural");
trad-n(x) ->
   outml("error traduciendo el numero");

trad-g(x) ->
   free(x) /
   outm("femenino y masculino");
trad-g(mas) ->
   / outm("masculino");
trad-g(fem) ->
   / outm("femenino");
trad-g(x) ->
   outml("error traduciendo el genero");

trad-ng(n,g) ->
   trad-n(n) outm(" , ")
   trad-g(g) /;
trad-ng(x,y) ->
   outml("(error traduciendo el genero y numero)");

lis-ng((sin.fem).(plu.fem).(sin.mas).(plu.mas).nil) ->;

¦
 LA SEMANTICA DE LAS PALABRAS INTRODUCIDAS ES:
¦

sem(x,<ver,0>,<i,<x,i>>) ->;
sem(x,<ver,1>,<i,<j,<x,i,j>>>) ->;
sem(x,<ver,2>,<i,<j,<k,<x,i,j,k>>>>) ->;
sem(x,<ver,prop>,x) ->;
sem(x,np,x) ->;
sem(x,y,x) ->;

¦
```

```
 TEST DE RECONOCIMIENTO DE LAS PALABRAS DE UNA FRASE
!

test-recon-pal(l,l') ->
   palab(l,l'')
   dec-opc-trp(l,l'',13)
   sal-fr-fin(l,13,l') ;

sal-fr-fin(l,l',l'') ->
   no-iguales(l,l') /
   outml("LA FRASE PARA ANALIZAR ES:")
   line sal-lista1(l') line
   test-recon-pal(l',l'');
sal-fr-fin(l,l',l') ->;

palab(nil,nil) -> /;
palab(x.y,l) ->
   numero-natural(x)
   /
   palab(y,l);
palab(x.y,l) ->
   <x,u,v,w>
   /
   palab(y,l);
palab(x.y,x.l) ->
   palab(y,l);

dec-opc-trp(l,nil,l) -> /;
dec-opc-trp(l,l'',l') ->
   line
   outml("ERROR EN LAS PALABRAS")
   line
   sal-lista1(l'')
   line
   outml("SELECCIONE UNA DE LAS OPCIONES PARA CADA PALABRA:")
   line sal-opc(l,l'',l');

sal-opc(l,l'',l') ->
   outml("1. ERROR ORTOGRAFICO (SERA SUSTITUIDA)")
   outml("2. NO DEFINIDA")
   outml("3. SUSTITUIR (POR MAS DE UNA PALABRA)")
   outml("4. ELIMINAR")
   outml("5. ACABAR EL PROCESO DE LA FRASE")
   line
   pal-op(l'',o)
   hacer-trp(o,l,l');

pal-op(nil,nil) -> /;
pal-op(x.l,<x,n>.l') ->
   line out(x)
   blancos(2) outm("OPCION") blancos(2)
   in-integer(n) dif(n,5) / line
   pal-op(l,l');
pal-op(l,l') -> fallo;

hacer-trp(nil,l,l) ->/;
hacer-trp(<x,1>.l'',l,l') ->/
```

```
   camb(x,l,l''')
   hacer-trp(l'',l''',l');
hacer-trp(<x,2>.l'',l,l') ->/
   outm("PALABRA PARA INTRODUCIR:    ") outl(x)
   anadir-dicc(x)
   hacer-trp(l'',l,l');
hacer-trp(<x,3>.l'',l,l') -> /
   camb'(x,l,l''')
   hacer-trp(l'',l''',l');
hacer-trp(<x,4>.l'',l,l') ->/
   quitar(x,l,l''')
   hacer-trp(l'',l''',l');
hacer-trp(<x,y>.l,l',l'') ->
   line outm("ERROR EN LA OPCION DE LA PALABRA ") outl(x)
   line
   hacer-trp(l,l',l'');

camb(x,l,l') ->
   line
   outm("LA PALABRA ") out(x) outm(" SE SUSTITUYE POR ")
   blancos(2)
   in-ident(y)
   sustituir(x,y,l,l')
   line;

camb'(x,l,l') ->
   line
   outm("LA PALABRA ") out(x) outm(" SE SUSTITUYE POR ")
   outml("(UNA LISTA ACABADA EN PUNTO)")
   blancos(2)
   in-sentence(y,s) filtro(y,z) quitar(".",z,z')
   line
   sustituir''(x,z',l,l');

!
```

```
 MECANISMO DE ANALISIS TOTAL
¦

frontera(x,l) ->
   hojas(x,nil,l);

hojas(x,l,l1) ->
  freeze(x,hojas'(x,l,l1));

hojas'(vacio,l,l) ->;
hojas'(<x>,l,x.l) ->;
hojas'(<x,x1>,l,l1) ->
   hojas(x1,l,l1);
hojas'(<x,x1,x2>,l,l1) ->
   hojas(x2,l,l1)
   detec(<x,x1,x2>);
hojas'(<x,x1,x2,x3>,l,l1) ->
   hojas(x3,l,l2)
   hojas(x2,l2,l1)
   detec(<x,x1>);
hojas'(<x,x1,x2,x3,x4>,l,l1) ->
   hojas(x4,l,l3)
   hojas(x3,l3,l2)
   hojas(x2,l2,l1)
   detec(<x,x1>);
hojas'(<x,x1,x2,x3,x4,x5>,l,l1) ->
   hojas(x5,l,l4)
   hojas(x4,l4,l3)
   hojas(x3,l3,l2)
   hojas(x2,l2,l1)
   detec(<x,x1>);
hojas'(<x,x1,x2,x3,x4,x5,x6>,l,l1) ->
   hojas(x6,l,l5)
   hojas(x5,l5,l4)
   hojas(x4,l4,l3)
   hojas(x3,l3,l2)
   hojas(x2,l2,l1)
   detec(<x,x1>);
hojas'(<x,x1,x2,x3,x4,x5,x6,x7>,l,l1) ->
   hojas(x7,l,l6)
   hojas(x6,l6,l5)
   hojas(x5,l5,l4)
   hojas(x4,l4,l3)
   hojas(x3,l3,l2)
   hojas(x2,l2,l1)
   detec(<x,x1>);

¦
```

```
DETECCION Y RECUPERACION DE ERRORES DE CONCORDANCIA

(hace comenzar la verificacion de gen y num)
!

detec-err-con(<l>) -> /;
detec-err-con(x) ->
   outml("FALLO EN DETEC-ERR")
   fallo;

!unif de gen y num (cada parte antes que el total)!

detec(<c,<t,l>>) ->
   free(l) /
   no-terminal(c);
detec(<c,<t,l.r>>) ->
   freeze(l,paso-const(l,r));
detec(<c,<t,l>,<t',y>>) ->
   terminal(c) /
   freeze(l,unif(t,t',<c,y>,l));
detec(<c,<t,l>,x>) -> /
   no-terminal(c);
detec(<c,<t>,<t',y>>) ->
   terminal(c);

paso-const(l,x) ->
   sep-corch(x,x1,x2)
   paso-const-corch(l',x1)
   paso-const'''(l'',x2,nil)
   conc-est(l',l'',l);

paso-const-corch(nil,nil) -> /;
paso-const-corch(l,<x>.r) ->
   paso-const-corch(l',r)
   conc(<x>.nil,l',l);

sep-corch(x,nil,x.nil) -> free(x) /;
sep-corch(<x>,x.nil,nil) -> free(x) /;
sep-corch(<sint-co(nil)>,nil,<sint-co(nil)>.nil) -> /;
sep-corch(<<sint-co(nil)>>,<sint-co(nil)>.nil,nil) -> /;
sep-corch(x.y,x1,x.x2) ->
   free(x) /
   sep-corch(y,x1,x2);
sep-corch(<x>.y,x.x1,x2) ->
   free(x) /
   sep-corch(y,x1,x2);
sep-corch(<sint-co(nil)>.y,x1,<sint-co(nil)>.x2) ->/
   sep-corch(y,x1,x2);
sep-corch(<<sint-co(nil)>>.y,<sint-co(nil)>.x1,x2) ->/
   sep-corch(y,x1,x2);

conc-est(nil,l.nil,l) ->/;
conc-est(l1,l2,<l>) ->
   conc-sin-nil(l1,l2,l);

conc-sin-nil(x.nil,nil,x) ->/;
conc-sin-nil(x.y,nil,x.l) ->
```

```
    / conc-sin-nil(y,nil,l);
conc-sin-nil(l1,x.nil,x.l) ->
    / conc-sin-nil(l1,nil,l);

paso-const'(l.nil,l.nil) ->/;
paso-const'(l,<<t,x>>.<<t',y>>.u) ->
    tipo-eti(t)
    tipo-eti(t') /
    const-err(<<t,x>>,<<t',y>>,<<t'',z>>)
    paso-const'(l,<<t'',z>>.u);
paso-const'(l,<<t,x>>.<y.z>.u) ->
    tipo-eti(t) /
    const-err(<<t,x>>,y,<<t'',v>>)
    paso-const'(l,<<<t'',v>>.z>.u);
paso-const'(l,<y.z>.<<t,x>>.u) ->
    tipo-eti(t) /
    const-err(y,<<t,x>>,<<t'',v>>)
    paso-const'(l,<<<t'',v>>.z>.u);
paso-const'(l,<x.y>.<z.s>.u) ->
    const-err(x,z,z')
    conc'(y,s,s')
    paso-const'(l,<z'.s'>.u);

paso-const''(l,<l1>.r,a) ->
    freeze(l1,paso-const'''(l,r,<l1>.a)) ;

paso-const'''(nil,nil,nil) ->/;
paso-const'''(l,nil,a) ->
    paso-const'(l,a)/;
paso-const'''(l,x.y,a) ->
    paso-const''(l,x.y,a);

const-err(<<sint-co,a>>,<<sint-co,b>>,<<sint-co,c>>) ->
    conc(a,b,c) /;
const-err(<<sint-co,a>>,<<x,b>>,<<x,c>>) ->
    conc(a,b,c) /;
const-err(<<x,a>>,<<sint-co,b>>,<<x,c>>) ->
    conc(a,b,c) /;
const-err(<<x,a>>,<<y,b>>,<<z,c>>) ->
    conc(a,b,c)
    con-ce(x,y,z);

tipo-eti(num)-> /;
tipo-eti(gen) ->/;
tipo-eti(num-gen) ->/;
tipo-eti(sint-co) -> /;

unif(t,t,x,<sint-co(x.nil)>) -> /;
unif(s.g,s'.g,x,<num(x.nil)>) ->
    bound(s)
    bound(s')
    dif(s,s')  /;
unif(s.g,s.g',x,<gen(x.nil)>) ->
    bound(g)
    bound(g')
    dif(g,g') /;
unif(s.g,s'.g',x,<num-gen(x.nil)>) ->
```

```
   dif(s,s')
   dif(g,g');

con-ce(x,x,x) -> /;
con-ce(error(num-gen),x,num-gen) -> /;
con-ce(x,error(num-gen),num-gen) -> /;
con-ce(x,y,num-gen) ->;

!
 RECUPERACION DE ERRORES DE CONCORDANCIA
!

concordancia(<sint-co(a)>,l,l) -> /;
concordancia(<<t,x>>,l,l') ->
   tipo-eti(t)
   / line outm("ERROR DE CONCORDANCIA EN EL ")
   trad-tipo-error(t)
   line concordancia'(<<t,x>>,l,l') ;
concordancia(<<t,x>>.r,l,l') ->
   tipo-eti(t)
   / concordancia(<<t,x>>,l,l'')
   concordancia(r,l'',l') ;
concordancia(<<sint-co(a)>.r>,l,l') ->
   / concordancia(r,l,l');
concordancia(<<<t,x>>.r>,l,l') ->
   tipo-eti(t)
   / line outml("ERROR DE CONCORDANCIA EN EL ")
   trad-tipo-error(t)
   line concordancia'(<<t,x>>,l,l'')
   concordancia(r,l'',l');
concordancia(<x.r>,l,l') ->
   concordancia(x,l,l'')
   concordancia(r,l'',l');

concordancia'(<<e,a>>,l,s) ->
   inver(a,nil,a')
   num-gen(e,n.g)
   reconst-frase(l,n.g,a',s)
   preg2(s)  /;
concordancia'(x,y,z) ->
   line
   outml(" NO HAY MAS POSIBILIDADES")
   outml(" PARA RESTAURAR ERROR DE GENERO Y-O NUMERO")
   fallo;

reconst-frase(l,n.g,e,l') ->
   sub-frase(n.g,e,e',e'')
   sust-dos-ados(l,e',e'',l');

sub-frase(n.g,nil,nil,nil) ->;
sub-frase(n.g,<c,<x>>.e,x.e',y.e'') ->
   buscar-re(<c,<x>>,y,n.g)
   sub-frase(n.g,e,e',e'');

sust-dos-ados(l,nil,nil,l) -> /;
sust-dos-ados(x.y.l,x.y.e,u.v.e',u.v.l') ->
   / sust-dos-ados(l,e,e',l');
```

```
sust-dos-ados(u.v.l,x.y.e,e',u.l') ->
   / sust-dos-ados(v.l,x.y.e,e',l');
sust-dos-ados(x.l,x.nil,u.nil,u.l) -> /;
sust-dos-ados(u.l,x.nil,e',u.l') ->
   sust-dos-ados(l,x.nil,e',l');

preg2(s) ->
   sal-lista1(s)
   line outml(" ES LA FRASE CORRECTA ?")
   line
   in-ident(t)
   dec2(t,s);

dec2(no,s) -> / fallo;
dec2(si,s) -> page line sal-lista1(s)/;
dec2(x,s) -> preg2(s);

trad-tipo-error(num) ->
   outml("numero");
trad-tipo-error(gen) ->
   outml("genero");
trad-tipo-error(num-gen) ->
   outml("genero Y numero");

num-gen(num,sin.fem) -> ;
num-gen(num,plu.fem) ->;
num-gen(num,sin.mas) -> ;
num-gen(num,plu.mas) ->;
num-gen(gen,sin.fem) ->;
num-gen(gen,sin.mas) ->;
num-gen(gen,plu.fem) ->;
num-gen(gen,plu.mas) ->;
num-gen(num-gen,sin.fem) ->;
num-gen(num-gen,sin.mas) -> ;
num-gen(num-gen,plu.fem) ->  ;
num-gen(num-gen,plu.mas) ->   ;

buscar-re(<x,<y>>,z,n.g) ->
   loc-re(x,y,l)
   pertenece(<n.g,z>,l) /;

loc-re(ver,y,l) ->
   <ver(x),l>
   pertenece(<c,y>,l) /;
loc-re(x,y,l) ->
   <x,l>
   pertenece(<c,y>,l);

;
```

```
!
 GRAMATICA

(pri indica no preposicion)
(dac denota grupo nominal distinto del vacio)

FRASES DECLARATIVAS CAUSALES Y COMPUESTAS
!

frase-declarativa(<<t,l>,fr-causal(<t,l.<l1>.<l2>>,x,y,z,u)>
                  ,o) ->
   ind-causal(<x',x>)
   frase-dec(<<t',l1>,y>,o1)
   ind-causal(<x',z>,o1,o2,o)
   frase-declarativa(<<t'',l2>,u>,o2);
frase-declarativa(<<t,l>,<r,<t,l.<l1>.<l2>>,x,y>>,o) ->
   frase-dec(<<t',l1>,x>,o1)
   resto-declarativa(<<t'',l2>,r,y>,o1,o);

resto-declarativa(<<t,l>,fr-causal,resto1(<t,l>,x,y)>,o1,o) ->
   ind-causal(<unit,x>,o1,o2,o)
   frase-declarativa(<<t,l>,y>,o2);
resto-declarativa(<<t,<sint-co(nil)>>,fr-dec,vacio>,o,o) ->;

frase-dec(<<t,l>,<r,<t,l.<l1>.<l2>>,x,y>>,o) ->
   frase-sim(<<t',l1>,x>,o1)
   resto-dec(<<t'',l2>,r,y>,o1,o);

resto-dec(<<t,<sint-co(nil)>>,fr-sim,vacio>,o,o) ->;
resto-dec(<<t,l>,fr-comp,resto2(<t,l>,x,y)>,o1,<c,o1,o2>) ->
   ind-comp(<x>,c)
   frase-dec(<<t',l>,y>,o2);
resto-dec(<<t,l>,fr-comp,resto2(<t,l.<l1>.<l2>>,x,y,z,u)>,
          o1,<c,o1,<c,o2,o3>>) ->
   ind-comp'(<x>)
   frase-comas(<<t',l1>,y>,c,o2)
   ind-comp(<z>,c)
   frase-sim(<<t'',l2>,u>,o3);
resto-dec(<<t,l>,fr-comp,resto2(<t,l>,x,y)>,
          o1,conj(o1,no(o2))) ->
   ind-neg'(<x>)
   frase-sim(<<t',l>,y>,o2);

frase-comas(<<t,l>,fr-cc(<t,l.<l1>.<l2>>,x,y)>,c,o) ->
   frase-sim(<<t',l1>,x>,o1)
   resto-comas(<<t'',l2>,y>,c,o1,o);

resto-comas(<<t,<sint-co(nil)>>,vacio>,c,o,o) ->;
resto-comas(<<t,l>,resto3(<t,l>,x,y)>,c,o1,<c,o1,o2>) ->
   ind-comp'(<x>)
   frase-comas(<<t,l>,y>,o2);

!
```

```
FRASES DECLARATIVAS SIMPLES
!

frase-sim(<<t,l>,<c,<t,l.l1.l2>,x,y>>,o) ->
   grupo-nom(<<pri.t,l1>,x>,<i,o1>,o)
   resto-sim(<<t,l2>,c,y>,<i,o1>);

resto-sim(<<t,l>,fr-sim-neg,resto4(<t,l.l1.l2>,x,y,z)>,
          <i,no(o)>) ->
   ind-neg(<x>)
   grupo-verbal(<<t,l1>,y>,<i,o1>)
   resto-neg(<<t,l2>,z>,<i,o1>,o);
resto-sim(<<t,l>,fr-sim-afir,resto4(<t,l>,x)>,<i,o>) ->
   grupo-verbal(<<t,l>,x>,<i,o>);

resto-neg(<<t,l>,resto(<t,l>,x,y)>,<i,o1>,dis(o1,o2)) ->
   ind-neg'(<x>)
   grupo-verbal(<<t,l>,y>,<i,o2>);
resto-neg(<<t,<sint-co(nil)>>,vacio>,<i,o>,o) ->;

!
```

FRASES INTERROGATIVAS

```
(no hay por respuestas demasiado generales frases
introducidas por "que" y verbo ser, estar, tener, haber)
!

frase-int(<<t,1>,fr-icq(<t,1.11.12>,x,y)>,quien(i,o)) ->
   pron-int(<<t,11>,x>)
   verbo(<<0.t,12>,y>,<i,o>);
frase-int(<<t,1>,fr-icq(<t,1.11.12>,x,y)>,quien(i,o)) ->
   pron-int(<<t,11>,x>)
   grupo-verbal(<<t,12>,y>,<i,o>);

frase-int(<<t,1>,fr-iq(<t,1.11.12>,x,y,z)>,que(j,o)) ->
   pron-rel(<x>)
   verbo(<<1.t,11>,y>,<i,<j,o1>>)
   grupo-nom(<<pri.t,12>,z>,<i,o1>,o);
frase-int(<<t,1>,fr-iq(<t,1.11.12.<13>>,x,y,z,u)>,que(j, o))->
   pron-rel(<x>)
   verbo(<<2.t,11>,y>,<i,<j,<k,o1>>>)
   grupo-nom(<<dac.t,12>,z>,<i,o2>,o)
   grupo-nom(<<dac.t',13>,u>,<k,o1>,o2);
frase-int(<<t,1>,fr-iq(<t,1.11.12>,x,y,z)>,que(j,o)) ->
   pron-rel(<x>)
   verbo(<<2.t,11>,y>,<i,<j,<k,o1>>>)
   grupo-nom(<<pri.t,12>,z>,<i,o1>,o);
frase-int(<<t,1>,fr-iq(<t,1.11.12>,x,y,z)>,que(i,o)) ->
   pron-rel(<x>)
   grupo-nom(<<pri.t',11>,y>,<i,o1>,o)
   grupo-verbal(<<t'',12>,z>,<i,o1>);

frase-int(<<t,1>,fr-id(<t,1.11.12.13>,x,y,z)>,si-no(o)) ->
   verbo(<<ser.t,11>,x>,n)
   grupo-nom(<<pri.t,12>,y>,<i,o1>,o)
   atri-s(<<t,13>,z>,<i,o1>);
frase-int(<<t,1>,fr-id(<t,1.11.12.13>,x,y,z)>,si-no(o)) ->
   verbo(<<tener.t,11>,x>,n)
   grupo-nom(<<pri.t,12>,y>,<i,o1>,o)
   complemento-t(<<t',13>,z>,<i,o1>);
frase-int(<<t,1>,fr-id(<t,1.11.12>,x,y)>,si-no(o)) ->
   verbo(<<haber.t,11>,x>,n)
   complemento-h(<<t',12>,y>,o);
frase-int(<<t,1>,fr-id(<t,1.11.12>,x,y)>,si-no(o)) ->
   grupo-verbal(<<t,11>,x>,<i,o1>)
   grupo-nom(<<dac.t,12>,y>,<i,o1>,o);
frase-int(<<t,1>,fr-id(<t,1>,x)>,si-no(o)) ->
   frase-declarativa(<<t,1>,x>,o);

frase-int(<<t,1>,fr-ic(<t,1.11.12.13>,x,y,z)>,
          cuantos(i,conj(o1,o2))) ->
   adver-cant(<<t,11>,x>)
   nombre-comun(<<t,12>,y>,<i,o1>)
   grupo-verbal(<<t,13>,z>,<i,o2>);

!
```

```
FRASES RELATIVAS
¦

frase-rel(<<t,<sint-co(nil)>>,fr-rel(vacio)>,<i,o>,o) ->;
frase-rel(<<t,1>,fr-rel(<t,1>,x,y)>,<i,conj(o,o1)>,o) ->
   pron-rel(<x>)
   grupo-verbal(<<t,1>,y>,<i,o1>);
frase-rel(<<sin.g,1>,fr-rel(<t,1.11.12.13>,x,y,z,u)>,
          <i,existe(j,conj(o,conj(o2,o1)))>,o) ->
   preposicion(<x>)
   cuyyo(<<t,11>,y>)
   nombre-comun'(<<t,12>,z>,<i,<j,o2>>)
   grupo-verbal(<<t,13>,u>,<j,o1>);
frase-rel(<<plu.g,1>,fr-rel(<plu.g',1.11.12.13>,x,y,z,u)>,
           <i,existe(j,conj(o,conj(o2,o1)))>,o) ->
   preposicion(<x>)
   cuyyo(<<plu.g',11>,y>)
   nombre-comun'(<<plu.g',12>,z>,<i,<j,o2>>)
   grupo-verbal(<<plu.g',13>,u>,<j,o1>);

¦
```

GRUPO NOMINAL

(con dac se evitan dos analisis de la misma frase)

```
¦

grupo-nom(<<c.t,<sint-co(nil)>>,gn(vacio)>,<i,o>,o) ->
   dif(c,dac);
grupo-nom(<<c.t,l>,gn(<t,l.11.12>,x,y,z)>,<i,o1>,o) ->
   preposicion(<x>)
   nombre-propio(<<t,11>,y>,i)
   frase-rel(<<t,12>,z>,<i,o>,o1);
grupo-nom(<<c.t,l>,gn(<t,l.11.12.13>,x,y,z)>,<i,o1>,o) ->
   grupo-art(<<c.t,11>,x>,<j,o2>,<i,o1>,o)
   nombre-comun(<<t,12>,y>,<j,o3>)
   frase-rel(<<t,13>,z>,<j,o2>,o3);
grupo-nom(<<c.t,l>,gn(<t,l.11.12.<13>>,x,y,z)>,<i,o1>,o) ->
   grupo-art(<<c.t,11>,x>,<j,o2>,<i,o1>,o3)
   nombre-comun'(<<t,12>,y>,<j,<k,o2>>)
   grupo-nom(<<dac.t',13>,z>,<k,o3>,o);
grupo-nom(<<c.t,l>,gn(<t,l.11.12.<13>.<14>>,x,y,z,u)>,
          <i,o1>,o) ->
   grupo-art(<<c.t,11>,x>,<j,o2>,<i,o1>,o3)
   nombre-comun''(<<t,12>,y>,<j,<k,<m,o2>>>)
   grupo-nom(<<dac.t',13>,z>,<k,o3>,o4)
   grupo-nom(<<dac.t'',14>,u>,<m,o4>,o);
grupo-nom(<<c.t,l>,gn(<t,l.11.12.13.14>,x,y,z,u)>,<i,o1>,o) ->
   grupo-art(<<c.t,11>,x>,<j,o2>,<i,o1>,o)
   nombre-comun(<<t,12>,y>,<j,o3>)
   adjetivo(<<t,13>,z>,<j,o3>,o4)
   dif(z,adj(vacio))
   frase-rel(<<t,14>,u>,<j,o2>,o4);
grupo-nom(<<c.t,l>,gn(<t,l.11.12.13.<14>>,x,y,z,u)>,
          <i,o1>,o) ->
   grupo-art(<<c.t,11>,x>,<j,o4>,<i,o1>,o3)
   nombre-comun'(<<t,12>,y>,<j,<k,o2>>)
   adjetivo(<<t,13>,z>,<j,o2>,o4)
   dif(z,adj(vacio))
   grupo-nom(<<dac.t',14>,u>,<k,o3>,o);

¦
```

GRUPO VERBAL

```
!

grupo-verbal(<<t,1>,gv(<t,1.11.12>,x,y)>,<i,o>) ->
   verbo(<<ser.t,11>,x>,n)
   atri-s(<<t,12>,y>,<i,o>);
grupo-verbal(<<t,1>,gv(<t,1.11.<12>.<13>>,x,y,z)>,<p,o>) ->
   verbo(<<prop.t,11>,x>,p)
   adjetivo(<<sin.mas,12>,y>,<p,p>,o2)
   grupo-nom(<<c.t',13>,z>,<p,o2>,o);
grupo-verbal(<<t,1>,gv(<t,1.11.<12>>,x,y)>,<i,o>) ->
   verbo(<<haber.t,11>,x>,n)
   complemento-h(<<t',12>,y>,<i,o>);
grupo-verbal(<<t,1>,gv(<t,1.11.<12>>,x,y)>,<i,o>) ->
   verbo(<<tener.t,11>,x>,n)
   complemento-t(<<t',12>,y>,<i,o>);
grupo-verbal(<<t,1>,gv(<t,1.11.12>,x,y)>,<i,o>) ->
   verbo(<<estar.t,11>,x>,n)
   atri-e(<<t,12>,y>,<i,o>);
grupo-verbal(<<t,1>,gv(<t,1>,x)>,<i,o>) ->
   verbo(<<0.t,1>,x>,<i,o>);
grupo-verbal(<<t,1>,gv(<t,1.11.<12>>,x,y)>,<i,o>) ->
   verbo(<<1.t,11>,x>,<i,<j,o1>>)
   grupo-nom(<<c.t',12>,y>,<j,o1>,o);
grupo-verbal(<<t,1>,gv(<t,1.11.<12>.<13>>,x,y,z)>,<i,o>) ->
   verbo(<<2.t,11>,x>,<i,<j,<k,o1>>>)
   grupo-nom(<<dac.t',12>,y>,<j,o1>,o2)
   grupo-nom(<<c'.t'',13>,z>,<k,o2>,o);

!
```

```
COMPLEMENTO (TENER Y HABER)
!

complemento-t(<<t,1>,comp(<t,1.11.12.13>,x,y,z)>,
                          <i,existe(j,conj(o2,cant(j,n)))>) ->
   num-card(<<t,11>,x>,n)
   nombre-comun'(<<t,12>,y>,<i,<j,o1>>)
   adjetivo(<<t,13>,z>,<j,o1>,o2);
complemento-t(<<t,1>,comp(<t,1.11.12>,x,y)>,
               <i,existe(j,o2)>) ->
   nombre-comun'(<<t,11>,x>,<i,<j,o1>>)
   adjetivo(<<t,12>,y>,<j,o1>,o2);
complemento-t(<<t,1>,comp(<t,1.11.12.<13>>,x,y,z)>,
               <i,existe(j,o)>) ->
   nombre-comun''(<<t,11>,x>,<i,<j,<k,o1>>>)
   adjetivo(<<t,12>,y>,<j,o1>,o2)
   grupo-nom(<<dac.t',13>,z>,<k,o2>,o);


complemento-h(<<t,1>,comp(<t,1.11.12.13>,x,y,z)>,
                          <i,existe(j,conj(o2,cant(j,n)))>) ->
   num-card(<<t,11>,x>,n)
   nombre-comun(<<t,12>,y>,<j,o1>)
   adjetivo(<<t,13>,z>,<j,o1>,o2);
complemento-h(<<t,1>,comp(<t,1.11.12>,x,y)>,
               <i,existe(j,o2)>) ->
   nombre-comun(<<t,11>,x>,<j,o1>)
   adjetivo(<<t,12>,y>,<j,o1>,o2);
complemento-h(<<t,1>,comp(<t,1.11.12.<13>>,x,y,z)>,
               <i,existe(j,o)>) ->
   nombre-comun'(<<t,11>,x>,<j,<k,o1>>)
   adjetivo(<<t,12>,y>,<j,o1>,o2)
   grupo-nom(<<dac.t',13>,z>,<k,o2>,o);
complemento-h(<<t,1>,comp(<t,1>,x)>,<i,o>) ->
   grupo-nom(<<pri.t',1>,x>,<j,vacio>,o);
!
```

```
ATRIBUTO (SER Y ESTAR)
¦

atri-s(<<t,1>,att(<t,1>,x)>,<i,o>) ->
   adjetivo(<<t,1>,x>,<i,o>,o)
   dif(x,adj(vacio));
atri-s(<<n.g,1>,att(<n.g,1.11.12.13>,x,y,z)>,<i,o>) ->
   articulo(<<n.g',11>,x>,a,b,c)
   nombre-comun(<<n.g',12>,y>,<i,o1>)
   adjetivo(<<n.g',13>,z>,<i,o1>,o);
atri-s(<<n.g,1>,att(<n.g,1.11.12.13.<14>>,x,y,z,u)>,<i,o>) ->
   articulo(<<n.g',11>,x>,a,b,c)
   nombre-comun'(<<n.g',12>,y>,<i,<j,o1>>)
   adjetivo(<<n.g',13>,z>,<i,o1>,o2)
   grupo-nom(<<dac.t'',14>,u>,<j,o2>,o);
atri-s(<<t,1>,att(<t,1.11.12.13>,x,y,z)>,<i,o>) ->
   articulo(<<t,11>,x>,a,b,c)
   nombre-comun(<<t,12>,y>,<i,o1>)
   frase-rel(<<t,13>,z>,<i,o>,o1);

atri-e(<<t,1>,att(<t,1>,x)>,<i,o>) ->
   adjetivo(<<t,1>,x>,<i,o>,o)
   dif(x,adj(vacio));
atri-e(<<t,1>,att(<t,1.11.12>,x,y)>,<i,o>) ->
   adjetivo'(<<t,11>,x>,<i,<j,o1>>,o1)
   grupo-nom(<<dac.t',12>,y>,<j,o1>,o)
   dif(x,adj(vacio));

¦
```

```
GRUPO DE ARTICULOS
¦

grupo-art(<<c.t,l>,gr-art-c(<t,l>,<sin.mas,<n,<x>>>)>,
                  <i,o1>,<i,o2>,existe(i,conj(o1,o2))) ->
   <x,gr-art-c,sin.mas,n>
   dif(c,pri);
grupo-art(<<c.t,l>,gr-art-c(<t,l>,<t',<prep,<de>>>)>,
                  <i,o1>,<i,o2>,conj(o1,o2)) ->
   dif(c,pri);
grupo-art(<<c.t,l>,gr-art(<t,l>,x,y)>,a,b,d) ->
   preposicion(<x>)
   articulo(<<t,l>,y>,a,b,d);
grupo-art(<<t,<sint-co(nil)>>,gr-art(vacio)>,
          <i,o1>,<i,o2>,conj(o1,o2)) ->;

¦
ARTICULO Y PREPOSICION
¦

articulo(<<t,l>,art-def(<t,l>,<sin.g,<x>>)>,
                  <i,o1>,<i,o2>,existe(i,conj(o1,o2))) ->
   <x,art-def,sin.g,y>;
articulo(<<t,l>,art-def(<t,l>,<plu.g,<x>>)>,
                  <i,o1>,<i,o2>,todo(i,dis(no(o1),o2))) ->
   <x,art-def,plu.g,y>;
articulo(<t,art-indef(t,<t',<x>>)>,
                  <i,o1>,<i,o2>,existe(i,conj(o1,o2))) ->
   <x,art-indef,t',y>;

preposicion(<prep(vacio)>) ->;
preposicion(<prep(<x>)>) ->
   <x,prep,t',y>;

¦
```

```
VERBOS
¦

verbo(<<c.t,l>,ver(<t,l>,<t',<x>>)>,y) ->
   <x,ver(c),t',y>;

¦
NOMBRE COMUN, PROPIO Y ADJETIVO
¦

nombre-propio(<<t,l>,np(<t,l>,<t',<x>>)>,y) ->
   <x,np,t',y>;

nombre-comun(<<t,l>,nc(<t,l>,<t',<x>>)>,<i,<y,i>>) ->
   <x,nc,t',y>;

nombre-comun'(<<t,l>,nc(<t,l>,<t',<x>>)>,<i,<j,<y,i,j>>>) ->
   <x,nc,t',y>;

nombre-comun''(<<t,l>,nc(<t,l>,<t',<x>>)>,
                 <i,<j,<k,<y,i,j,k>>>>) ->
   <x,nc,t',y>;

adjetivo(<<t,l>,adj(<t,l>,<t',<x>>)>,<i,<a,i>>,<a,i>) ->
   <x,adj,t',a>;
adjetivo(<<t,l>,adj(<t,l>,<t',<x>>)>,<i,o1>,conj(o1,<a,i>)) ->
   <x,adj,t',a>;
adjetivo(<<t,<sint-co(nil)>>,adj(vacio)>,<i,o>,o) ->;

adjetivo'(<<t,l>,adj(<t,l>,<t',<x>>)>,
          <i,<j,<a,i,j>>>,<a,i,j>) ->
   <x,adj,t',a>;

¦
PRONOMBRES Y ADVERBIOS
¦

cuyyo(<<t,l>,pron-cuyo(<t,l>,<t',<x>>)>) ->
   <x,pron-cuyo,t',y>;

pron-int(<<t,l>,pron-quien(<t,l>,<t',<x>>)>) ->
   <x,pron-quien,t',y>;

adver-cant(<<t,l>,adv-cant(<t,l>,<t',<x>>)>) ->
   <x,adv-cant,t',y>;

pron-rel(<pron-rel(<x>)>) ->
   <x,pron-rel,t,s> /;
<pron-rel,categ-term> ->;

¦
```

```
CONECTORES
¦

ind-comp(<part-dc(<x>)>,y) ->
   <x,part-dc,t',y>;
ind-comp'(<part-cc(<cc>)>) ->;

ind-neg(<part-neg(<x>)>) ->
   <x,part-neg,t',y>;
ind-neg'(<part-neg'(<x>)>) ->
   <x,part-neg',t',y>;

ind-causal(<si-ent,adv(<si>)>) ->;
ind-causal(<bas-para,adv(<basta>)>) ->;
ind-causal(<para,prep(<para>)>) ->;
ind-causal(<si-ent,adv(<entonces>)>,x,y,imp(x,y)) ->;
ind-causal(<si-ent,part-cc(<cc>)>,x,y,imp(x,y)) ->;
ind-causal(<bas-para,prep(<para>)>,x,y,imp(x,y)) ->;
ind-causal(<para,es-nec-que(s,ser(<es>),adv(<necesario>),
          pron-rel(<que>))>,x,y,imp(x,y)) ->;
ind-causal(<para,es-suf-que(s,ser(<es>),adv(<suficiente>),
          pron-rel(<que>))>,x,y,imp(y,x)) ->;
ind-causal(<unit,solo-si(s,adv(<solo>),adv(<si>))>,
           x,y,imp(x,y)) ->;
ind-causal(<unit,suf-para(s,adv(<suficiente>),prep(<para>))>,
          x,y,imp(x,y)) ->;
ind-causal(<unit,amq(s,prep(<aa>),adv(<menos>),
                        pron-rel(<que>))>,
           x,y,imp(no(x),y)) ->;
ind-causal(<unit,adv(<si>)>,x,y,imp(y,x)) ->;
ind-causal(<unit,nec-para(s,adv(<necesario>),prep(<para>))>,
          x,y,imp(y,x)) ->;
ind-causal(<unit,sii(s,adv(<si>),part-dc(<yy>),adv(<solo>),
                        adv(<si>))>,
           x,y,conj(imp(x,y),imp(y,x))) ->;
ind-causal(<unit,suf-nec(s,ser(<es>),adv(<suficiente>),
                           part-dc(<yy>),
                           adv(<necesario>),prep(<para>))>,
           x,y,conj(imp(x,y),imp(y,x))) ->;

¦
```

```
CARDINALES
¦

num-card(<<t,l>,card(<t,l>,<t',<x>>)>,y) ->
          <x,card,t',y>;
<card,categ-term>->;

¦



ACCESO A LAS PALABRAS DEL DICCIONARIO
¦

palabra(nc,sin.fem,palabra) ->;
palabra(x) ->
   <x,c,n.g,s>
   outl(c);

palabras(nc,plu.fem,palabra) ->;
palabras(c) ->
   <c,l>
   sal-pal(l);

sal-pal(nil) ->;
sal-pal(<n.g,x>.l) ->
   outl(x)
   sal-pal(l);

¦
```

```
ARTICULOS
¦

<el,art-def,sin.mas,art> ->;
<la,art-def,sin.fem,art> ->;
<los,art-def,plu.mas,art> ->;
<las,art-def,plu.fem,art> ->;

<un,art-indef,sin.mas,art> ->;
<un,card,sin.mas,1> ->;
<una,art-indef,sin.fem,art> ->;
<una,card,sin.fem,1> ->;
<unos,art-indef,plu.mas,art> ->;
<unas,art-indef,plu.fem,art> ->;

<del,gr-art-c,t,con1> ->;
<al,gr-art-c,t,con2> ->;

<gr-art-c,categ-term> ->;

¦
PRONOMBRES
¦

que(pron-rel,t',r) ->;

<cuyo,pron-cuyo,sin.mas,x> ->;
<cuyos,pron-cuyo,plu.mas,x> ->;
<cuya,pron-cuyo,sin.fem,x> ->;
<cuyas,pron-cuyo,plu.fem,x> ->;

<cual,pron-cual,sin.g,r> ->;
<cuales,pron-cual,plu.g,r> ->;

quien(pron-quien,sin.g,r) ->;
<quienes,pron-quien,plu.g,pr> ->;

¦
PREPOSICIONES
¦

<prep,categ-term> ->;

<de,prep,t,pp> ->;
<en,prep,t,pp> ->;
<aa,prep,t,pp> ->;
<para,prep,t,pp> ->;

¦
```

```
ADVERVIOS Y CONECTORES
!

<adv,categ-term> ->;
<part-neg,categ-term> ->;
<part-neg,categ-term> ->;
<part-dc,categ-term> ->;
<part-cc,categ-term> ->;

<yy,part-dc,t,conj> ->;
<pero,part-dc,t,conj> ->;
<aunque,part-dc,t,conj> ->;
<oo,part-dc,t,dis> ->;

<cc,part-cc,t,coma> ->;

no(part-neg,t,neg) ->;
<ni,part-neg',t,neg> ->;

<si,adv,t,caus> ->;
<entonces,adv,t,caus> ->;
<basta,adv,t,caus> ->;
<necesario,adv,t,caus> ->;
<suficiente,adv,t,caus> ->;
<solo,adv,t,caus> ->;
<menos,adv,t,caus> ->;

<cuantos,adv-cant,plu.mas,x> ->;
<cuantas,adv-cant,plu.fem,x> ->;
<cuanto,adv-cant,sin.mas,x> ->;
<cuanta,adv-cant,sin.fem,x> ->;

!
```

```
VERBOS SER ESTAR TENER Y HABER
¦

<es,ver(ser),sin.g,s> ->;
<son,ver(ser),plu.g,s> ->;

<tiene,ver(tener),sin.g,r> ->;
<tienen,ver(tener),plu.g,r> ->;

<esta,ver(estar),sin.g,e> ->;
<estan,ver(estar),plu.g,e> ->;

<hay,ver(haber),sin.g,h> ->;

¦
NUMERALES CARDINALES
¦

<uno,card,sin.mas,1> ->;
<dos,card,plu.g,2> ->;
<tres,card,plu.g,3> ->;
<cuatro,card,plu.g,4> ->;
<cinco,card,plu.g,5> ->;
<seis,card,plu.g,6> ->;
<siete,card,plu.g,7> ->;
<ocho,card,plu.g,8> ->;
<nueve,card,plu.g,9> ->;
<cero,card,plu.g,0> ->;

¦
```

```
DEFINICION AGRUPADA
¦

art-indef(<sin.fen,una>.<plu.fem,las>.
          <sin.mas,un>.<plu.mas,unos>.nil) ->;
<art-indef,categ-term> ->;

art-def(<sin.fem,la>.<plu.fem,las>.<sin.mas,el>.<plu.mas,los>
         .nil) ->;
<art-def,categ-term> ->;

pron-cuyo(<sin.mas,cuyo>.<plu.mas,cuyos>.
          <sin.fem,cuya>.<plu.fem,cuyas>.nil) ->;
<pron-cuyo,categ-term> ->;

adv-cant(<plu.mas,cuantos>.<plu.fem,cuantas>.nil) ->;
<adv-cant,categ-term> ->;

pron-cual(<sin.g,cual>.<plu.g,cuales>.nil) ->;
<pron-cual,categ-term> ->;

pron-quien(<sin.g,quien>.<plu.g,quienes>.nil) ->;
<pron-quien,categ-term> ->;

¦
PARA INTRODUCCION DE PALABRAS
¦

grupos-sintacticos(np.nc.adj.ver.nil) ->;

                ;
```

```
;
DEFINICIONES AUXILIARES
;

numero-natural(x) ->
   integer(x) /;
numero-natural(x) ->
   fallo;

sustituir(y,z,nil,nil) ->;
sustituir(y,z,y.l,z.l') ->
   sustituir(y,z,l,l');
sustituir(y,z,u.l,u.l') ->
   dif(y,u)
   sustituir(y,z,l,l');

sustituir'(nil,nil,l,l') ->;
sustituir'(x.e,y.s,l,l') ->
   sustituir(x,y,l,l'')
   sustituir'(e,s,l'',l');

sustituir''(x,s,x.l,l') ->
   / pon''(s,l,l');
sustituir''(x,s,y.l,y.l') ->
   sustituir''(x,s,l,l');

pon''(nil,l,l) ->;
pon''(x.y,l,x.l') ->
   pon''(y,l,l');

terminal(x) ->
   <x,categ-term> / ;
terminal(x) -> fallo;

no-terminal(x) ->
   terminal(x) / fallo;
no-terminal(x) ->;

lista(nil) -> /;
lista(x.y) ->
   lista(y);

pertenece(x,nil) ->
   / fallo;
pertenece(x,x.l) -> /;
pertenece(x,y.l) ->
   pertenece(x,l);

inver(nil,l,l) ->/;
inver(x.l,l',s) ->
   inver(l,x.l',s);

elemento(x,x.l) ->;
elemento(x,y.l) ->
   dif(x,y)
   elemento(x,l);
```

```
quitar(x,nil,l) ->
   / fallo;
quitar(x,x.l,l) -> /;
quitar(x,y.l,y.l') ->
   dif(x,y)
   quitar(x,l,l');

quitar-todo(x,nil,nil) ->;
quitar-todo(x,x.l,l') ->
   quitar-todo(x,l,l');
quitar-todo(x,y.l,y.l') ->
   dif(x,y)
   quitar-todo(x,l,l');

conc(nil,x,x) ->/;
conc(x.l,y,x.l') ->
   conc(l,y,l');

conc'(x.y,l,x.l') ->
   conc'(y,l,l');
conc'(x,l,x.l) -> ;

hacer-lista(<x.y>,l,r) ->
   hacer-lista(y,l,r')
   hacer-lista(x,r',r) /;
hacer-lista(x.y,l,r) ->
   hacer-lista(y,l,r')
   hacer-lista(x,r',r) /;
hacer-lista(<x>,l,x.l) ->;

sal-lista2(nil) ->;
sal-lista2(x.y) ->
   outl(x)
   sal-lista2(y);

tiempo->
   cpu-time(x) outm("TIEMPO DE CPU = ")
   outl(x) line;

blancos(0) ->/;
blancos(x) ->
   outm(" ")
   val(sub(x,1),y) blancos(y);

no-iguales(x.l,nil) ->/;
no-iguales(nil,x.l) ->/;
no-iguales(nil,nil) -> / fallo;
no-iguales(x.l,x.l') -> / no-iguales(l,l');
no-iguales(x.l,y.l') ->;

sal-lista1(nil) -> line;
sal-lista1(x.y) ->
   out(x) outm(" ")
   sal-lista1(y);

!
```

SALIDA DE ESTRUCTURA SINTACTICA DE FRASES

```
(oculta la estructura indicadora de la concordancia
y los nodos resto (sas: sal arbol sintactico))
!

dibuja-arbol-sint(x,y) ->
   sas(x,y)
   outl(y);

sas(x,x) ->
   free(x) / ;
sas(<c,<t,l>,x,y,z,u>,<c,x',y',z',u'>) ->
   / sas(x,x')
   sas(y,y')
   sas(z,z')
   sas(u,u');
sas(<c,<t,l>,x,y,z,u,v>,<c,x',y',z',u',v'>) ->
   / sas(x,x')
   sas(y,y')
   sas(z,z')
   sas(u,u')
   sas(v,v');
sas(<c,<t,l>,x,y,z,u,v,w>,<c,x',y',z',u',v',w'>) ->
   / sas(x,x')
   sas(y,y')
   sas(z,z')
   sas(u,u')
   sas(v,v')
   sas(w,w');
sas(<c,<t,l>,x,y,vacio>,<c,x',y'>) ->
   / sas(x,x')
   sas(y,y');
sas(<c,<t,l>,x,y,z>,<c,x',y',z'>) ->
   / sas(x,x')
   sas(y,y')
   sas(z,z');
sas(<c,<t,l>,x,vacio>,<c,x'>) ->
   / sas(x,x');
sas(<c,<t,l>,x,resto1(<t',l'>,a,b)>,<c,x',a',b'>) ->
   / sas(x,x')
   sas(a,a')
   sas(b,b');
sas(<c,<t,l>,x,resto2(<t',l'>,a,b)>,<c,x',a',b'>) ->
   / sas(x,x')
   sas(a,a')
   sas(b,b');
sas(<c,<t,l>,x,resto2(<t',l'>,a,b,e,d)>,<c,x',a',b',e',d'>) ->
   / sas(x,x')
   sas(a,a')
   sas(b,b')
   sas(e,e')
   sas(d,d');
sas(<c,<t,l>,x,resto3(<t',l'>,a,b)>,<c,x',a',b'>) ->
   / sas(x,x')
   sas(a,a')
   sas(b,b');
```

```
sas(<c,<t,l>,x,resto4(<t',l'>,a)>,<c,x',a'>) ->
   / sas(x,x')
   sas(a,a');
sas(<c,<t,l>,x,resto4(<t',l'>,a,b,vacio)>,
          <c,x',a',b'>)->
   / sas(x,x')
   sas(a,a')
   sas(b,b');
sas(<c,<t,l>,x,resto4(<t',l'>,a,b,resto(<t'',l''>,e,d))>,
          <c,x',a',b',e',d'>)->
   / sas(x,x')
   sas(a,a')
   sas(b,b')
   sas(e,e')
   sas(d,d');
sas(<c,<t,l>,x,y>,<c,x',y'>) ->
   / sas(x,x')
   sas(y,y');
sas(<c,<t,l>,<t',x>>,<c,x'>) ->
   / sas(x,x');
sas(<c,<t,l>,x>,<c,x'>) ->
   /sas(x,x');
sas(<c,<t>,<t',x>>,<c,x'>) ->
   / sas(x,x');
sas(<x,y>,<x,y'>) ->
   / sas(y,y');
sas(c,c) ->;

                ;
```

```
!
SALIDA CON MODULO DESSIN
DEL RESULTADO DEL ANALISIS
!

dib-frase ->
   paper
   outml(">dib-frase;")
   frase(t,l,i,o) line
   outm("FRASE ") stipo(t) line
   sal-frase(t,l) line
   outml("SINTAXIS:") line
   draw-tree(i)
   outml("SEMANTICA:") line
         draw-tree(o) line;

sal-frase(dec,l) -> sal-fr(dec,l);
sal-frase(int,l) -> outm("¿ ") sal-fr(int,l);

sal-fr(dec,nil) -> outml(".");
sal-fr(int,nil) -> outml("?");
sal-fr(t,x.y) ->
   out(x) outm(" ")
   sal-fr(t,y);

stipo(dec) -> outm("DECLARATIVA:") line;
stipo(int) ->
   outm("INTERROGATIVA:") line;

;
```

```
¦
DICCIONARIO

Es particular al dominio de la aplicacion
¦

          ...

np(<sin.mas,Juan>.nil) ->;
np(<sin.fem,Alicia>.nil) ->;
np(<sin.mas,Antonio>.nil) ->;
np(<sin.mas,Manuel>.nil) ->;
np(categ-term) ->;

          ...

nc(<sin.fem,hermana>.<plu.fem,hermanas>.
   <sin.mas,hermano>.<plu.mas,hermanos>.nil) ->;
nc(<sin.mas,amigo>.<sin.fem,amiga>.
   <plu.fem,amigas>.<plu.mas,amigos>.nil)->;
nc(<sin.mas,tren>.<plu.mas,trenes>.nil) ->;
nc(<sin.fem,ciudad>.<plu.fem,ciudades>.nil) ->;
nc(<sin.mas,acceso>.<plu.mas,accesos>.nil) ->;
nc(<sin.mas,problema>.<plu.mas,problemas>.nil) ->;
nc(categ-term) ->;

          ...

adj(<sin.x246,feliz>.<plu.x246,felices>.nil) ->;
adj(<sin.mas,contento>.<sin.fem,contenta>.<plu.fem,contentas>.
    <plu.mas,contentos>.nil) ->;
adj(<sin.mas,arriesgado>.<plu.mas,arriesgados>.nil) ->;
adj(<sin.mas,concedido>.<sin.fem,concedida>.
    <plu.fem,concedidas>.<plu.mas,concedidos>.nil) ->;
adj(<sin.mas,bueno>.<sin.fem,buena>.<plu.fem,buenas>.
     <plu.mas,buenos>.nil) ->;
 adj(<sin.x567,favorable>.<plu.x567,favorables>.nil) ->;
adj(categ-term) ->;

          ...

<ver(1),<plu.x563,estudian>.<sin.x563,estudia>.nil> ->;
<ver(1),<plu.x497,miran>.<sin.x497,mira>.nil> ->;
<ver(2),<sin.x790,da>.<plu.x790,dan>.nil> ->;
<ver(2),<sin.x853,pide>.<plu.x853,piden>.nil> ->;
<ver(2),<sin.x1640,concede>.<plu.x1640,conceden>.nil> ->;
<ver(0),<plu.x1587,vienen>.<sin.x1587,viene>.nil> ->;
<ver(1),<sin.x656,sonrie>.<plu.x656,sonrien>.nil> ->;
<ver(2),<sin.x278,ama>.<plu.x278,aman>.nil> ->;
<ver(prop),<sin.x,llueve>.nil> ->;
<ver(ser),<sin.g,es>.<plu.g,son>.nil> ->;
<ver(tener),<sin.g,tiene>.<plu.g,tienen>.nil> ->;
<ver(haber),<sin.g,hay>.nil> ->;
<ver(estar),<sin.g,esta>.<plu.g,estan>.nil> ->;
ver(categ-term) ->;
¦
```

```
;
<coyuntura,nc,sin.fem,coyuntura> ->;

<coyunturas,nc,plu.fem,coyuntura> ->;

<peticion,nc,sin.fem,peticion> ->;

<peticiones,nc,plu.fem,peticion> ->;

<credito,nc,sin.mas,credito> ->;

<creditos,nc,plu.mas,credito> ->;

<expansiva,adj,sin.fem,expansiva> ->;

<expansivas,adj,plu.fem,expansiva> ->;

<expansivo,adj,sin.mas,exp> ->;

<expansivos,adj,plu.mas,exp> ->;

<estable,adj,sin.g,estable> ->;

<estables,adj,plu.g,estable> ->;

<vienen,ver(0),plu.x1587,<i,<viene,i>>> ->;

<viene,ver(0),sin.x1587,<i,<viene,i>>> ->;

<ama,ver(2),sin.x278,<i,<j,<k,<ama,i,j,k>>>>> ->;

<aman,ver(2),plu.x278,<i,<j,<k,<aman,i,j,k>>>>> ->;

<llueve,ver(prop),sin.mas,llueve> ->;

<Antonio,np,sin.mas,Antonio> ->;

<Alicia,np,sin.fem,Alicia> ->;

<sonrien,ver(1),plu.x1587,<i,<j,<sonrie,i,j>>>> ->;

<sonrie,ver(1),sin.x1587,<i,<j,<sonrie,i,j>>>> ->;

        ...
 ; End world: Dic
```

CURRICULUM VITAE DE LA AUTORA

Licenciatura en Ciencias Matemáticas sección Computación, en el periodo 1976-1981.

Obtención del Grado de Licenciatura en Matemáticas en Junio de 1983 con tesina "Reescritura de Términos en Lenguajes de Especificación" dirigida por el Profesor Fernando Orejas de la Facultad de Informática de Barcelona.

Realización de los cursos de Doctorado en La Facultad de Informática de Madrid durante los periodos académicos 1983-1984 y 1984-1985.

EXPERIENCIA DOCENTE EN UNIVERSIDAD

Cursos 1981-1982 y 1982-1983 : Encargada de Curso de la asignatura "Fundamentos Matemáticos de la Informática (Teoria de Automatas y Teoria de Grafos)" en el Departamento de Informática Teórica de la Facultad de Informática de San Sebastian (Universidad del Pais Vasco).

Cursos 1983-1987 : Encargada de curso en la asignatura Lógica Informática, impartiendo además en La asignatura de Inteligencia Artificial el Lenguaje de Programación Prolog, todo ello en el Departamento de Lógica e Inteligencia Artificial de la Facultad de Informática de Madrid.

PARTICIPACION EN PROYECTOS DE INVESTIGACION

Curso 1982-83: Proyecto CAPRA (Sistema experto de enseñanza de conceptos básicos de Programación). Dirigido por Felisa Verdejo. Facultad de Informática de San Sebastián.

Cursos 1984-86: Proyecto Director Ideal (Estudio sodre el rendimiento y calidad de gestión de las tecnicas de inteligencia Artificial aplicadas a la gestión bancaria).

Dirigido por José Cuena. Facultad de Informática de Madrid.

Cursos 1986- : Proyecto ESTELA (Estudios de entornos lógicos y sus aplicaciones). Dirigido por José Cuena. Facultad de Informática de Madrid.

ESTANCIAS DE INVESTIGACION EN EL EXTRANJERO

Julio 1984 y 1985: Realización de un trabajo de Tratamiento del Lenguaje Natural (el castellano) con el formalismo de Gramáticas de Metamorfosis (A. Colmerauer) con el Grupo de Inteligencia Artificial de Marsella. Dirigido por R. Pasero Profesor de la Universidad de Luminy Marsella del Grupo de Inteligencia Artificial .

PUBLICACIONES O COMUNICACIONES PRESENTADAS EN CONGRESOS

Enero 1983: Comunicación presentada en las III Jornadas Hispano-Francesas de Informática Teórica y Programación celebradas en Montpellier, con título "Reescritura de términos condicional" con F. Orejas y M.L. Navarro.

Junio 1985: En las "Jornadas de Inteligencia Artificial" organizadas por la Universidad Complutense en Madrid presentación con título "Programación lógica en la Comprensión del Lenguaje Natural". (Trabajos presentados publicados por Texas Instruments España).

Octubre 1985: En la reunión anual de la Asociación Española de Informatica y Automática (AEIA), celebradas en Madrid, comunicación presentada con J. Cuena con Título: "La deducción natural como motor de inferencia para sistemas expertos proposicionales". Trabajos presentados publicados por AEIA).

Diciembre 1985: En la Revista de la Asociación Española para

la Inteligencia Artificial (AEPIA) del Periodo Primavera Verano 86 (Resumen de la reunion anual celebrada en Madrid) con título: "Crítica del Lenguaje de Programación Prolog", en colaboración con J. Garcia San-Luis y A. Salmerón.

Mayo 1987: Comunicación presentada en "7th International Workshop on Expert Systems and their Applications" con título: "An Expert System for Financial Risk Evaluation and Decision Making" en colaboración con J. Cuena. Avignon 77.

Enero 1988: Comunicación presentada en IBERAMIA 88 con título: "El proyecto Director Ideal: Sistema experto para la evaluación y toma de decisiones en riesgos financieros", en colaboración con M. Alonso, J. Cuena y R. Gimeno de la Facultad de Informática de Madrid.

DIRECCION DE TRABAJOS FIN DE ESTUDIOS

1986 : Tesis de Master en Informática "Un interprete de frases en lenguaje natural como comandos SEQUEL". Realizado por D. Alberto Pazmiño Proaño en la Facultad de Informática de Madrid.

1987: Proyecto Fin de Carrera de Informática: "SELENA: Un sistema de entendimiento del lenguaje natural para el ajedrez". Realizado por Jose Luis Fernandez.

OTROS MERITOS

1987: Traducción al castellano del libro "Prolog" de F. Giannesini, H. Kanoui, R. Pasero, M. Van Caneghem en colaboración con Jose Cuena. Trabajo encargado por la editorial Addison-Wesley.